NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EXPLORING FIELDS WITH SHIFT REGISTERS

by

Jody L. Radowicz

September 2006

Co-Thesis Advisors: George Dinolt
Harold Fredricksen

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 2006	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Exploring Fields with Shift Registers	5. FUNDING NUMBERS
6. AUTHOR(S) Jody L. Radowicz	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

The S-Boxes used in the AES algorithm are generated by field extensions of the Galois field over two elements, called GF(2). Therefore, understanding the field extensions provides a method of analysis, potentially efficient implementation, and efficient attacks. Different polynomials can be used to generate the fields, and we explore the set of polynomials $x^2 + x + \alpha^j$ over GF(2^n) where α is a primitive element of GF(2^n).

The results of this work are the first steps towards a full understanding of the field that AES computation occurs in—GF(2^8). The charts created with the data we gathered detail which power of the current primitive root is equal to previous primitive roots for fields up through GF(2^{16}) created by polynomials of the form $x^2 + x + \alpha^i$ for a primitive element α. Currently, a C++ program will also provide all the primitive polynomials of the form $x^2 + x + \alpha^i$ for a primitive element α over the fields through GF(2^{32}). This work also led to a deeper understanding of certain elements of a field and their equivalent shift register state. In addition, given an irreducible polynomial $f(x) = x^2 + \alpha^i x + \alpha^j$ over GF(2^n), the period (and therefore the primitivity) can be determined by a new theorem without running the shift register generated by $f(x)$.

14. SUBJECT TERMS Field, Galois Shift Register, Primitive, Field Extensions, Exponential Algorithm	15. NUMBER OF PAGES 99
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EXPLORING FIELDS WITH SHIFT REGISTERS

Jody L. Radowicz
Civilian, Federal Cyber Corps
B.S., North Central College, 2002
M.S., University of Michigan, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2006

Author: Jody L. Radowicz

Approved by: Dr. George Dinolt
Co-Thesis Advisor

Dr. Harold Fredricksen
Co-Thesis Advisor

Dr. Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The S-Boxes used in the AES algorithm are generated by field extensions of the Galois field over two elements, called GF(2). Therefore, understanding the field extensions provides a method of analysis, potentially efficient implementation, and efficient attacks. Different polynomials can be used to generate the fields, and we explore the set of polynomials $x^2 + x + \alpha^j$ over GF(2^n) where α is a primitive element of GF(2^n).

The results of this work are the first steps towards a full understanding of the field that AES computation occurs in—GF(2^8). The charts created with the data we gathered detail which power of the current primitive root is equal to previous primitive roots for fields up through GF(2^{16}) created by polynomials of the form $x^2 + x + \alpha^i$ for a primitive element α. Currently, a C++ program will also provide all the primitive polynomials of the form $x^2 + x + \alpha^i$ for a primitive element α over the fields through GF(2^{32}). This work also led to a deeper understanding of certain elements of a field and their equivalent shift register state. In addition, given an irreducible polynomial $f(x) = x^2 + \alpha^i x + \alpha^j$ over GF(2^n), the period (and therefore the primitivity) can be determined by a new theorem without running the shift register generated by $f(x)$.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my advisors, Dr. Harold Fredricksen and Dr. George Dinolt, for the time, effort, and guidance they have provided throughout this project.

This material is based upon work supported by the National Science Foundation under Grant No. DUE0414102. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The S-Boxes used in the AES algorithm are generated by field extensions of the Galois field over two elements, called GF(2). Therefore, understanding the field extensions provides a method of analysis, potentially efficient implementation, and efficient attacks. Different polynomials can be used to generate the fields—the AES implementation uses one set, Canright [1] uses another, Conway [2] uses another way, and we explore the set of polynomials $x^2 + x + \alpha^j$ over GF(2^n) where α is a primitive element of GF(2^n). In particular, we look at the structure of the constant coefficients of the polynomials.

A primitive element of a Galois field of size p^n is an element whose powers are all different. Since there are $p^n - 1$ of these powers, these powers actually exhaust all of the nonzero elements of the field. By definition of a field extension, the field that is being extended is a subfield of the larger field. So, elements that are in the subfield are also in the extension field. For example, for a primitive element m of the extension field and for each of the elements s in the subfield, there exists a power e of m so that $s = m^e$.

Suppose $x^2 + x + \alpha^j$ is a polynomial over the field GF(2^n) with α being a primitive element of GF(2^n). Using an algorithm different from the typical algorithm for building fields with Galois shift registers, we are able to show whether or not the polynomial is irreducible (i.e., can be factored) over that field. With a new theorem, we are able to determine whether or not the polynomial is primitive when it is also irreducible. In addition, we use the alternate algorithm to discover information about the primitive roots from previous fields in the hopes that this will further our understanding of the fields built by these particular polynomials.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

In order to understand the algorithms and methods presented in this paper, we need to review some mathematical concepts as well as other topics including linear shift registers.

A. FIELD THEORY REVIEW

We need to first review some definitions and results from abstract algebra. These results can be found in any standard algebra text such as Dummit and Foote's Abstract Algebra [3] or Gallian's Contemporary Abstract Algebra [4].

1. Groups

A *group G* is a set of elements with a binary operation defined on those elements that has the following properties:

1. The binary operation is closed over the group, meaning that the binary operation performed on any two elements of the group will result in another element of the group.
2. The binary operation is associative.
3. There exists an identity element for the operation.
4. Each element of the group has an inverse for the operation.

An example of a group where the binary operation is addition modulo *n* is the set of integers $0,1,2,\ldots,n-1$, denoted $\mathbb{Z}_n$. In this group, 0 is the identity, and $n-k$ is the inverse of *k*.

2. Rings and Ideals

A *ring R* is a set with 2 binary operations + and $\times$, called addition and multiplication, such that the following properties hold:

1. (R, +) is a commutative group.
2. Multiplication is associative.
3. The distributive laws hold in R: for all a, b, c in R:
   $(a+b)\times c = (a\times c)+(b\times c)$ and $a\times(b+c) = (a\times b)+(a\times c)$.

A *subring S* of a ring *R* is a subset of *R* that is also a ring with the operations of *R*. A subring *A* of a ring *R* is an *ideal* of *R* if for all *r* in *R* and for all *a* in *A*, *ra* and *ar* are in *A*. In other words, *A* absorbs the elements from *R*. A ring *R* is a *commutative ring with*

unity if multiplication is commutative and there exists a multiplicative identity in *R*. Suppose *R* is a commutative ring with unity. Let *a* be an element in *R*. Then the set $\langle a \rangle = \{ra \mid r \in R\}$ is an ideal of *R* called a *principal ideal*.

Let *R* be a ring and let *A* be a subring of *R*. If *R* is a commutative ring, then $R[x] = \{a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 \mid a_i \in R\}$ is the *ring of polynomials over R*.

Theorem 1: If *A* is an ideal, then we may form the *factor ring* $R / A = \{r + A \mid r \in R\}$. In this case, the set of *cosets* $\{r + A \mid r \in R\}$ is a ring under the operations:

1. $(s + A) + (t + A) = (s + t) + A$ and
2. $(s + A)(t + A) = (st) + A$ for *s* and *t* in *R*.

For example, let $\mathbb{R}[x]$ be the ring of polynomials whose coefficients are real numbers. Let $\langle x^2 + 1 \rangle$ be the principal ideal generated by $x^2 + 1$. So, $\langle x^2 + 1 \rangle = \{f(x)(x^2 + 1) \mid f(x) \in \mathbb{R}[x]\}$. Then, the factor ring $\mathbb{R}[x] / \langle x^2 + 1 \rangle = \{g(x) + \langle x^2 + 1 \rangle \mid g(x) \in \mathbb{R}[x]\}$. Now, since $g(x)$ is in $\mathbb{R}[x]$, $g(x)$ may be written as $g(x) = q(x)(x^2 + 1) + r(x)$ where the degree of $r(x)$ is less than the degree of $x^2 + 1$ by the division algorithm. So, $r(x) = ax + b$ for some *a* and *b* in the real numbers. Therefore,

$$\begin{aligned}
\mathbb{R}[x] / \langle x^2 + 1 \rangle &= \{g(x) + \langle x^2 + 1 \rangle \mid g(x) \in \mathbb{R}[x]\} \\
&= \{q(x)(x^2 + 1) + r(x) + \langle x^2 + 1 \rangle\} \\
&= \{r(x) + \langle x^2 + 1 \rangle \mid r(x) \in \mathbb{R}[x]\} \text{ because the ideal } \langle x^2 + 1 \rangle \text{ absorbs the} \\
&\qquad\qquad \text{term } q(x)(x^2 + 1) \\
&= \{ax + b + \langle x^2 + 1 \rangle \mid a, b \in \mathbb{R}\} \text{ by definition of } r(x).
\end{aligned}$$

The notation can be simplified by denoting a coset $ax + b + \langle x^2 + 1 \rangle$ by its *coset representative* $ax + b$.

An ideal *A* of a ring *R* is a *proper ideal* of *R* if *A* is a proper subset of *R*. A proper ideal *A* of *R* is a *maximal ideal* of *R* if, whenever *B* is an ideal of *R* and $A \subseteq B \subseteq R$, then $B = A$ or $B = R$.

3. Fields

A *field F* is a set of elements with two binary operations + and •, usually called addition and multiplication, defined on it that has specific properties:

1. $(F,+)$ is a commutative group with identity 0.
2. $(F - \{0\}, \bullet)$ is a commutative group with identity 1.
3. The distributive law holds for all a, b, c in F:

$$a\bullet(b+c) = (a\bullet b) + (a\bullet c).$$

Familiar fields include the rational numbers, the real numbers, and the complex numbers. However, we are interested in fields with only a finite number of elements, referred to as *finite fields*.

An example of a finite field with 2 elements is the set {0, 1}. Addition and multiplication in this field are defined as follows:

0 + 0 = 0	0 * 0 = 0
0 + 1 = 1	0 * 1 = 0
1 + 0 = 1	1 * 0 = 0
1 + 1 = 0	1 * 1 = 1

Table 1. Multiplication and Addition Rules in the Field {0, 1}

For this field, (logical) XOR is the addition operation and (logical) AND is the multiplication operation.

Theorem 2: Finite fields have only a prime or prime power number of elements.

The fields with a prime number of elements are represented by the integers mod p, for any prime p. Addition and multiplication are done modulo p. A finite field that has p^n elements for a prime p and any positive integer n is called a *Galois field*, denoted GF(p^n).

In a field, the group of nonzero elements is *cyclic*, meaning that there is at least one element whose powers exhaust all of the nonzero elements of the field. The *order* of an element α of a group is the smallest positive integer n such that $\alpha^n = 1$.

Theorem 3: The order of an element in a group also divides the number of elements in a group.

Theorem 4: Let R be a commutative ring with unity and let A be an ideal of R. Then R/A is a field if and only if A is a maximal ideal.

4. Constructing a Field of p^n Elements

A *polynomial over a particular field* F is a polynomial $a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$ such that each coefficient a_i is an element of the field F. A polynomial $f(x)$ over a field F is *irreducible* if $f(x)$ cannot be factored as a product of two polynomials, both defined over F and both of degree lower than $f(x)$. Otherwise, $f(x)$ is *reducible*.

Theorem 5: Let F be a field and let $p(x)$ be in $F[x]$. Then $\langle p(x) \rangle$ is a maximal ideal in $F[x]$ if and only if $p(x)$ is irreducible over F.

Theorem 6: If $p(x)$ is an irreducible polynomial, then $F[x]/\langle p(x) \rangle$ is a field.

In other words, to create a field that has a prime power p^m of elements, we need an irreducible polynomial $f(x)$ of degree m over the prime field GF(p).

For example, consider $\mathbb{Z}_2 = \{0,1\}$ and the polynomial $f(x) = x^3 + x + 1$. If $f(x)$ were reducible, it would have at least one factor of degree one. This would imply that $f(x)$ would have a root in $\mathbb{Z}_2$. But $f(0) = f(1) = 1$ implies that there are no roots of $f(x)$ in $\mathbb{Z}_2$. So, $f(x)$ is irreducible. Therefore, $\mathbb{Z}_2[x]/\langle f(x) \rangle$ is a field. And $\mathbb{Z}_2[x]/\langle x^3 + x + 1 \rangle = \{ax^2 + bx + c + \langle x^3 + x + 1 \rangle \mid a, b, c \in \mathbb{Z}_2\}$ is a field of $2^3 = 8$ elements. If we designate a coset by its coset representative, then the elements of the field are $\{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$.

5. Different Representations of Elements of a Field

There are actually several different ways to represent the elements of a finite field. Above is an example of creating a field from the factor ring $F[x]/\langle p(x)\rangle$ where F is a field and $p(x)$ is an irreducible polynomial. If the degree of $p(x)$ is m and F has p^q elements, then the field $F[x]/\langle p(x)\rangle$ is called a *degree m extension field of F*. The elements of the larger field can be expressed as m-tuples chosen from the field F.

Theorem 7: Let $F[x]/\langle p(x)\rangle$ be an extension field such that F is a field and $p(x)$ is a degree m irreducible polynomial. Then the elements of the extension field are isomorphic to the polynomials of degree less than m over F.

Theorem 8: If F is a field and $p(x)$ is an irreducible polynomial over F, then there exists a field K containing an isomorphic copy of F in which $p(x)$ has a root.

In other words, there exists an extension field K of F in which $p(x)$ has a root.

The *order of a polynomial* $f(x)$ is the smallest integer n such that $f(x)$ divides $x^n - 1$. A *primitive polynomial* is an irreducible polynomial $f(x)$ of degree m over GF(p) such that the smallest n for which $f(x)$ divides $x^n - 1$ is $n = p^m - 1$. For example, consider $f(x) = x^2 + x + 1$ over GF(2). Note that $f(0) = 1$ and $f(1) = 1$. So, there are no roots of $f(x)$ in GF(2). Therefore, $f(x)$ is irreducible over GF(2). Note also that addition and subtraction are the same over GF(2). Then,

$$\frac{x^1 + 1}{x^2 + x + 1} = \frac{x^1 + 1}{x^2 + x + 1}$$

$$\frac{x^2 + 1}{x^2 + x + 1} = 1 + \frac{x}{x^2 + x + 1}$$

$$\frac{x^3 + 1}{x^2 + x + 1} = x + 1$$

So, the smallest integer n such that $x^2 + x + 1$ divides $x^n + 1$ is 3. Therefore, the order of $f(x)$ is 3 and it is primitive.

When creating the field GF(p^m) from GF(p) and an irreducible polynomial $f(x)$ of degree m, if $f(x)$ is not primitive, then multiplication is as follows:

$$a(x) \bullet b(x) = c(x) \text{ (modulo } p, \text{ modulo } f(x))$$

where we reduce the product both modulo p and modulo the irreducible polynomial $f(x)$.

However, multiplication can be accomplished much more easily if the irreducible polynomial $f(x)$ is also primitive.

Theorem 9: If the polynomial $f(x)$ is primitive, then a root α of the polynomial $f(x)$ is also *primitive*, meaning that the powers of α exhaust the nonzero elements of the field.

Theorem 10: There is always a primitive element of the field with which we can perform multiplication in this convenient way.

For example, let $f(x) = x^4 + x + 1$ be a polynomial over GF(2). Note that $f(0) = f(1) = 1$. So, $f(x)$ has no roots in GF(2) and therefore does not contain a degree 1 polynomial as a factor. However, it could still factor into two degree 2 polynomials. The only possible degree 2 polynomial that $f(x)$ could factor into that does not itself factor into two degree 1 polynomials is $x^2 + x + 1$. But when $f(x)$ is divided by this polynomial, a remainder of 1 results. So, $f(x)$ does not factor into two degree 2 polynomials, and is therefore irreducible. Suppose that α is a root of $f(x)$. Then, we can represent GF(2^4) as a set of polynomials in α of degree less than 4. However, if we find a primitive element in GF(2^4), we can also represent the nonzero elements of the field as powers of that primitive element. In this case, α happens to be primitive, and we can create a table that will simplify both addition and multiplication operations in the field.

We verify that the order of α is $2^4 - 1 = 15$, and that α is primitive.

$$\begin{aligned}
\alpha^{15} &= (\alpha^5)^3 = (\alpha \bullet \alpha^4)^3 = (\alpha \bullet (\alpha + 1))^3 = (\alpha^2 + \alpha)^3 \\
&= ((\alpha^2 + \alpha)(\alpha^2 + \alpha))(\alpha^2 + \alpha) = (\alpha^4 + \alpha^2)(\alpha^2 + \alpha) \\
&= \alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 = \alpha^4 \bullet \alpha^2 + \alpha^4 \bullet \alpha + (\alpha + 1) + \alpha^3 \\
&= (\alpha + 1)\alpha^2 + (\alpha + 1)\alpha + (\alpha + 1) + \alpha^3 \\
&= \alpha^3 + \alpha^2 + \alpha^2 + \alpha + \alpha + 1 + \alpha^3 = 1
\end{aligned}$$

So, the order of α divides 15 and could be either 3, 5, or 15. However, $\alpha^3 \neq 1$ and $\alpha^5 = \alpha^4 \bullet \alpha = (\alpha + 1)\alpha = \alpha^2 + \alpha \neq 1$. Therefore, the order of α is 15 and it is primitive. Now we can create a table of the two different representations of each element of the field – one representation as a polynomial in α of degree less than 4 and the other as a power of α. In this case, multiplication can now be defined by $\alpha^i \bullet \alpha^j = \alpha^{(i+j) \bmod (2^4 - 1)}$.

Element as a power of α	Element as a polynomial in α
α^0	1
α^1	α
α^2	α^2
α^3	α^3
α^4	$\alpha + 1$
α^5	$\alpha^2 + \alpha$
α^6	$\alpha^3 + \alpha^2$
α^7	$\alpha^3 + \alpha + 1$
α^8	$\alpha^2 + 1$
α^9	$\alpha^3 + \alpha$
α^{10}	$\alpha^2 + \alpha + 1$
α^{11}	$\alpha^3 + \alpha^2 + \alpha$
α^{12}	$\alpha^3 + \alpha^2 + \alpha + 1$
α^{13}	$\alpha^3 + \alpha^2 + 1$
α^{14}	$\alpha^3 + 1$

Table 2. Table of the Two Representations of the Elements of the Field GF(2^4)

6. Building Fields with Different Extensions

One can build fields with different irreducible polynomials as well as different degree extensions.

Theorem 11: Fields of order p^n are all isomorphic to GF(p^n).

For example, the field GF(2^8) can be built with three degree 2 extensions, one degree 4 extension followed by a degree 2 extension, or one degree 8 extension.

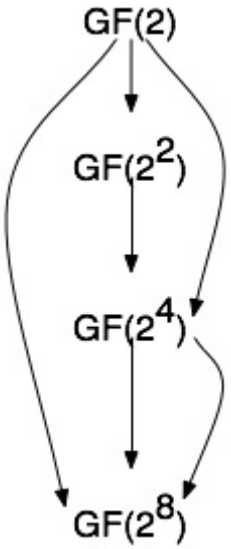


Figure 1. Different Extensions from GF(2) to GF(2^8)

First, we show the field GF(2^8) being built with three degree 2 extensions. Consider the polynomial $f(x) = x^2 + x + 1$ over GF(2). As we saw above, $f(x)$ is irreducible and primitive. Therefore, $GF(2)/\langle x^2 + x + 1 \rangle$ is a field of 4 elements and isomorphic to GF(2^2). Suppose a is a root of $f(x)$. The order of a is 3, and a is primitive. All of the nonzero elements of GF(2^2) can be expressed as powers of a as we showed earlier. Now consider the field GF(2^2) and the polynomial $g(x) = x^2 + x + a$. Since $g(0) = g(1) = a$ and $g(a) = g(a^2) = a^2$, $g(x)$ is irreducible. Therefore, $GF(2^2)/\langle x^2 + x + a \rangle$ is isomorphic to GF(2^4). Let b be a root of $g(x)$. Since b is a root of $g(x)$,

$g(b) = b^2 + b + a = 0$ which implies $b^2 = b + a$. In Table 2.3, we show that the order of b is 15. So, b is primitive and each nonzero element of GF(2^4) can be written as a power of b. The table of the powers of b is below.

b^0	1
b^1	b
b^2	b+a
b^3	a^2b+a
b^4	b+1
b^5	a
b^6	ab
b^7	ab+a^2
b^8	b+a^2
b^9	ab+a
b^{10}	a^2
b^{11}	a^2b
b^{12}	a^2b+1
b^{13}	ab+1
b^{14}	a^2b+a^2

Table 3. Table of the Two Representations of the Elements of the Field GF(2^4)

Note that $b^5 = a$. So even though a is an element of the smaller field GF(2^2), there is a copy of it (as well as a^2 and $a^3 = 1$) in the bigger field GF(2^4).

Consider the polynomial $h(x) = x^2 + x + b^7$ over GF(2^4). It can be shown that no elements of GF(2^4) are roots of $h(x)$ similar to the way that we showed that $f(x)$ has no

roots in GF(2). So, $h(x)$ cannot be factored and is irreducible. Therefore, $GF(2^4)/\langle h(x)\rangle$ is isomorphic to the field GF(2^8). Note that for this paper, we are mainly interested in degree 2 extensions of the form $x^2 + x + \alpha^i$ for some primitive element α. In this case, we use a to denote a primitive element in GF(2^2), b to denote a primitive element in GF(2^4), c to denote a primitive element in GF(2^8), d to denote a primitive element in GF(2^{16}) and so on.

Now, we can also build GF(2^8) from GF(2) using a degree 4 extension followed by a degree 2 extension. For example, take GF(2) and the polynomial $s(x) = x^4 + x + 1$. We have already shown that $s(x)$ is irreducible and that the root α of $s(x)$ is primitive. So, the field $GF(2)/\langle s(x)\rangle$ is isomorphic to GF(2^4). Consider the polynomial $t(x) = x^2 + x + \alpha^{11}$ over GF(2^4). Again, it can be shown that $t(x)$ is irreducible over GF(2^4) by showing that there are no roots of $t(x)$ in GF(2^4) and that therefore, the polynomial cannot be factored. So, $GF(2^4)/\langle t(x)\rangle$ is a field of 2^8 elements and is isomorphic to GF(2^8).

Since 2^8 is a power of a prime, we can also build GF(2^8) directly from GF(2) with just one extension of degree 8. Consider the polynomial $v(x) = x^8 + x^4 + x^3 + x + 1$ over GF(2). Now $v(0) = v(1) = 1$, so there are no roots of $v(x)$ in GF(2). Therefore, $v(x)$ cannot be factored into any degree 1 polynomials. The only irreducible degree 2 polynomial over GF(2) is $x^2 + x + 1$, and the remainder when $v(x)$ is divided by it is $x + 1$. So, $v(x)$ is not divisible by any degree 2 polynomials that do not themselves factor. There are two degree 3 irreducible polynomials over GF(2)— $x^3 + x + 1$ and $x^3 + x^2 + 1$. However, when $v(x)$ is divided by each of them, the remainders are $x + 1$ and x^2, respectively. Thus, $v(x)$ is not divisible by any degree 3 irreducible polynomials over GF(2). Now, $x^4 + x^3 + x^2 + x + 1$, $x^4 + x^3 + 1$, and $x^4 + x + 1$ are the only degree 4 irreducible polynomials over GF(2), and the remainders are $x^3 + x^2$, $x^3 + x^2$, and $x^3 + x^2 + 1$, respectively, when $v(x)$ is divided by each of the polynomials. There is no need to check any other degrees. For example, if $v(x)$ was divisible by a degree 5 polynomial, then it

must be divisible by a degree 3 polynomial. Therefore, $v(x)$ cannot be factored and is irreducible. So, $GF(2)/\langle v(x)\rangle$ is a field of order 2^8 and is isomorphic to GF(2^8).

7. Conjugates

Let β be an element of GF(p^m). The *conjugates* of β with respect to GF(p) are $\beta, \beta^p, \beta^{p^2}, \beta^{p^3}, \ldots$. The set of conjugates of β form the *conjugacy class* of β.

Theorem 12: The conjugacy class of β in GF(p^m) contains d elements, where d is the smallest integer such that $\beta^{p^d} = \beta$.

For example, consider GF(2^3) and let $\alpha \neq 1$ be a nonzero element in GF(2^3). The conjugacy class of α is $\{\alpha, (\alpha)^2 = \alpha^2, (\alpha)^{2^2} = \alpha^4, (\alpha)^{2^3} = \alpha^1\} = \{\alpha, \alpha^2, \alpha^4\}$. The conjugacy class of 0 is $\{0\}$ and the conjugacy class of 1 is $\{1\}$.

Theorem 13: Let $f(x)$ be a primitive polynomial over a field, and let α be a root of $f(x)$. Then, the roots of $f(x)$ are exactly the conjugates of α.

Theorem 14: If elements are in the same conjugacy class, then they have the same order.

B. LINEAR FEEDBACK SHIFT REGISTERS (LFSR)

Linear feedback shift registers are an important tool that can be used to build the fields GF(2^n). Golomb's Shift Register Sequences [5] is a good reference for linear feedback shift registers. Fellin's *Primitive Shift Registers* [6] is also a good quick introduction.

1. An Overview of LFSR's

A binary *shift register* of span n is a set of n storage elements, each holding either a 0 or a 1. The content of the n storage elements is the *state* of the register at a particular time. A *feedback function* is also associated with the shift register. When a new bit is needed, each bit in the register at a particular time is shifted in the direction of the increasing index at the next time until the feedback function determines the bit in the lowest-order element. Let s_i be the contents of the ith storage element at a particular time for a shift register with n storage elements. In general, if the feedback function at time t is $f(s_0, \ldots, s_{n-1}) = c_0 s_0 + \ldots + c_{n-2} s_{n-2} + c_{n-1} s_{n-1} = s_0$ a time $t + 1$ for $c_i \in \{0,1\}$ where addition

is performed modulo 2, then the shift register is a *linear feedback shift register* (because s_0 is a linear function of the other s_i's). The output tap is s_{n-1}. The shift register is completely dependent on its previous state and the feedback function. So, once the state returns to its initial state, we know exactly what the sequence of next states of the register will be. The *period* of a shift register is the length of the output sequence before the sequence starts to repeat.

Theorem 15: The period is at most $2^n - 1$ where n is the number of registers in the LFSR since the all 0 state cannot appear on a cycle which includes 1s.

Note that if the register is initialized with $s_i = 0$ for all i, the output sequence would be 00000... .

2. Galois Shift Register

An example of a LFSR is the *Galois shift register*. Instead of the general feedback function described above, the contents of the storage elements are XOR'ed together based on the design of the particular Galois shift register [7]. This design is explained in the section below.

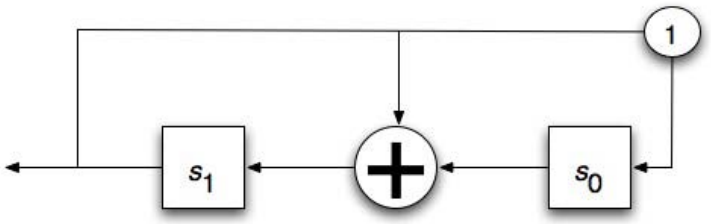


Figure 2. Galois Shift Register Generated by $f(x) = x^2 + x + 1$

If we initialize the contents of the storage elements with 0 1, the states of the Galois shift register are:

	s_1	s_0
time 0	0	1
time 1	1	0
time 2	1	1
time 3	0	1
⋮	⋮	⋮

Table 4. States of the Galois Shift Register

and are calculated by the following rules:

new s_0 = 1 • old s_1

new s_1 = old s_1 + old s_0 (modulo 2)

The output sequence of this shift register is 011011011… .Galois shift registers are very useful for creating fields since there exists a mapping of a state to the nonzero elements of a field.

3. Polynomial Associated with LFSR

By definition, the *characteristic polynomial* of the sequence of bits that make up the contents of the n registers at time t and of the shift register itself is $f(x) = x^n + \sum_{i=0}^{n-1} c_i x^i$, where the c_i's are the feedback function coefficients. This polynomial generates the LFSR. Consider the polynomial $g(x) = x^n + v_{n-1}x^{n-1} + v_{n-2}x^{n-2} + \ldots + v_1 x + v_0$. Then, the Galois shift register it generates is below.

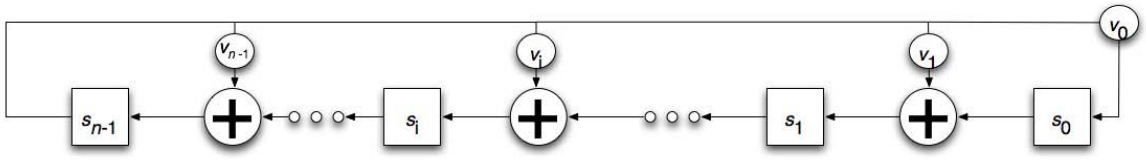


Figure 3. Generic Galois Shift Register

4. How Galois LFSRs Can be Used to Build Fields

We can build all of the nonzero elements of a field with Galois shift registers. For example, recall the primitive polynomial $f(x) = x^2 + x + 1$ over GF(2). Let a be one root of $f(x)$ in the field $GF(2^2) = GF(2)/\langle f(x) \rangle$. Each element of GF($2^2$) can be written as $s \bullet a^1 + t \bullet a^0$ for s and t in GF(2). So there will be 2 storage elements in the shift register. One storage element holds the coefficient of a^1 and the other holds the coefficient of a^0. Next, we determine how $f(x)$ affects the feedback, which is the coefficient of a^2. But $a^2 = a + 1$ in this field. So, the feedback goes to the registers that hold the coefficients of the a^1 and a^0 terms, i.e., s^1 and s^0, respectively.

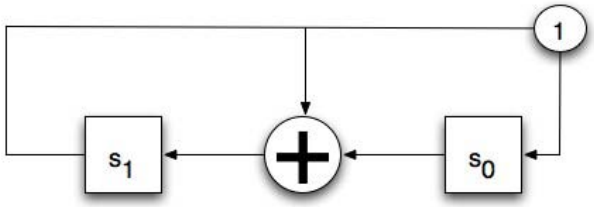


Figure 4. Galois Shift Register Generated by $f(x) = x^2 + x + 1$

Each subsequent step of the shift register is equivalent to multiplying the element of the field (which is equivalent to the current state of the shift register) by a and then reducing that result modulo $f(x)$. This occurs because shifting the contents of the registers is equivalent to multiplication by a and XOR'ing the coefficients is equivalent to reducing modulo 2.

To see this, look at the table of states for the shift register generated by $f(x)$ below.

	power of a	contents of registers	equivalent polynomial in a
time 0	a^0	0 1	$0 \bullet a + 1 \bullet 1 = 1$
time 1	a^1	1 0	$1 \bullet a + 0 \bullet 1 = a$
time 2	a^2	1 1	$1 \bullet a + 1 \bullet 1 = a + 1$

Table 5. Contents of Galois Shift Register and Equivalent Field Elements

Note that the state of the register at time 2 verifies the relationship $a^2 = a + 1$, which is equivalent to the fact that *a* is a root of the polynomial *f*(*x*).

Now, we used primitive polynomials to build the shift register, but we could have just as easily used an irreducible polynomial that was not primitive to build the shift register. However, a root of an imprimitive polynomial is not primitive, and therefore the powers of the root will not exhaust all the nonzero elements of the field. Since the result of each step of the register is equivalent to multiplying the current element by the root of the imprimitive polynomial used to build the shift register, the states of the shift register will not result in all of the nonzero elements of the field appearing. All the nonzero elements of the field appearing are equivalent to all the different states of a Galois shift register appearing. This happens only if the polynomial used to create the register is primitive.

For example, consider the polynomial $h(x) = x^2 + x + b^3$ over GF(2^4) = $\{0, b, b^2, b^3, b^4, b^5, b^6, b^7, b^8, b^9, b^{10}, b^{11}, b^{12}, b^{13}, b^{14}, b^{15}\}$. Now, *h*(*x*) has no roots in GF(2^4), so it is irreducible. The Galois shift register generated by *h*(*x*) is below.

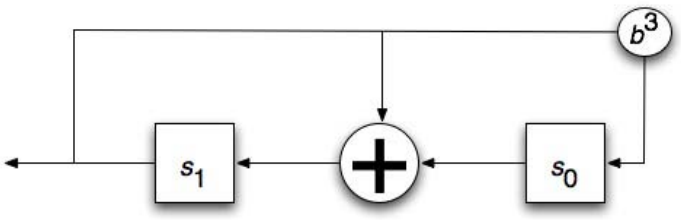


Figure 5. Galois Shift Register Generated by $x^2 + x + b^3$

If we initialize the contents of the storage elements with 0 1, the states of the Galois shift register are:

	s_1	s_0
time 0	0	1
time 1	1	0
time 2	1	b^3
time 3	b^{14}	b^3
⋮	⋮	⋮
time 85	0	1

Table 6. States of the Galois Shift Register Generated by $x^2 + x + b^3$

and are calculated by the following rules:

new $s_0 = b^3 \bullet$ old s_1

new s_1 = old s_1 + old s_0 (modulo 2)

Since the period of this shift register is 85, the polynomial $h(x) = x^2 + x + b^3$ is irreducible but not primitive over GF(2^4).

III. MATH TOOLS

In order to gather data about primitive polynomials of the form $x^2 + x + \alpha^i$, we wrote two programs. One program, written in Mathematica, used the Standard Algorithm for Galois shift registers explained in Chapter II. Because computations are done in the field using built-in functions as well as a special library [8], the program runs slowly. As a result, we took a new approach and looked at the Exponential Algorithm for building fields with Galois shift registers. We programmed this new approach in C++. However, the Exponential Algorithm requires the Galois shift register table for every previous extension field. So, this C++ program needs a lot of memory, and it does not take long for a typical 32-bit x86 based machine to be insufficient. On the other hand, the C++ program is far superior to the Mathematica program in terms of its runtime.

A. MATHEMATICA PROGRAM INSIGHTS

In order to determine whether or not polynomials of the form $x^2 + x + \alpha^i$ are primitive over a given field and also to determine which power of the current primitive root is equal to previous primitive roots, we programmed a Galois shift register in Mathematica. When this particular shift register is run, addition and multiplication of the elements in the field are still being performed. So, for the Standard Algorithm for the Galois shift register as described in Chapter II and above, we still need computational algebra software. With the help of a Galois theory library for Mathematica [8], the algorithm is simple. Also, we minimized the work that the program did via the procedures explained below. The output of the program is just the states of the shift register, and an example is in Appendix A.

1. Existence of Irreducible Polynomials

The first step to finding primitive polynomials of the form $x^2 + x + \alpha^i$ for some primitive element α in GF(2^n) and some positive integer *i* over a particular field GF(2^n) is to find irreducible polynomials of that form. So, we minimized the work that the program did by ignoring reducible polynomials of the form $x^2 + x + \alpha^i$ over each field.

We can determine if irreducible polynomials of the form $x^2 + x + \alpha^i$ exist over a particular field by using a counting argument. However, since we know the total number of polynomials of the form $x^2 + x + \alpha^i$, we can also find the number of reducible polynomials of that form and then subtract.

For example, consider polynomials over GF(2^2). We know all degree 2 monic polynomials over the field are of the form $x^2 + ax + b$ where *a* and *b* are in GF(2^2). Since there are four choices for each of *a* and *b*, that leaves us with 16 different monic degree 2 polynomials over the field. Now, we need to determine which ones are irreducible and which ones are reducible. Suppose that $x^2 + ax + b$ is reducible. Then the polynomial factors into two degree 1 terms. In other words, $x^2 + ax + b = (x + s)(x + t)$ for some *s* and *t* in GF(2^2). We know all the possible values for *s* and *t*, so we can create a table of all the possible products of $(x + s)$ and $(x + t)$.

		s			
	*	0	1	α	α^2
	0	x^2	$x^2 + x$	$x^2 + \alpha x$	$x^2 + \alpha^2 x$
t	1	$x^2 + x$	$x^2 + 1$	$x^2 + \alpha^2 x + \alpha$	$x^2 + \alpha x + \alpha^2$
	α	$x^2 + \alpha x$	$x^2 + \alpha^2 x + \alpha$	$x^2 + \alpha^2$	$x^2 + x + 1$
	α^2	$x^2 + \alpha^2 x$	$x^2 + \alpha x + \alpha^2$	$x^2 + x + 1$	$x^2 + \alpha$

Table 7. Multiplication Table of All Possible Products of $(x + s)$ and $(x + t)$.

After taking all of the possible products of $(x + s)$ and $(x + t)$, there are only 10 different monic degree 2 polynomials. So, x^2, $x^2 + x$, $x^2 + 1$, $x^2 + \alpha x$, $x^2 + \alpha^2 x + \alpha$, $x^2 + \alpha^2$, $x^2 + \alpha^2 x$, $x^2 + \alpha x + \alpha^2$, $x^2 + x + 1$, and $x^2 + \alpha$ are the only polynomials that can be factored into $(x + s)(x + t)$ for some *s* and *t* in GF(2^2). This means that there are 16 – 10 = 6 monic degree 2 polynomials that cannot be factored. In other words, they are irreducible.

2. Irreducible Polynomials over a Field

If we look at the problem from a different angle, we can program the search for reducible polynomials. Suppose $f(x) = x^2 + x + \alpha^i$ is reducible and that α^j and α^k are roots. Then, $f(x) = x^2 + x + \alpha^i = (x + \alpha^j)(x + \alpha^k) = x^2 + (\alpha^j + \alpha^k)x + \alpha^{j+k}$. This only happens if $\alpha^j + \alpha^k = 1$. Once we find a pair (j, k) for which relationship holds, we know that the polynomial $x^2 + x + \alpha^{j+k}$ is reducible. Thus, we do not need to run the shift register generated by the polynomial to test if it is primitive.

3. Testing One Root Per Conjugacy Class

We know that elements in the same conjugacy class have the same order. So, if $x^2 + x + \alpha^i$ is a primitive polynomial over GF(2^n), then so is $x^2 + x + (\alpha^i)^{2^k}$ for every $(\alpha^i)^{2^k}$ in the conjugacy class of α^i. Therefore, we only need to run the shift register generated by $x^2 + x + \alpha^i$ to determine the periods of the polynomials whose constant coefficients are in the conjugacy class of α^i.

4. Mathematica Program Pseudocode

For example, the pseudocode for the Mathematica Program that builds GF(2^{16}) is below and the actual code is in Appendix B.

```
Pseudocode for building GF(2^16):
set directory to look for finite field library
declare the field extension GF(2)
declare the field extension GF(2^2)
declare the field extension GF(2^4)
declare the field extension GF(2^8)

open the outfile
```

```
set the variables lowerField //name given to the second to last declared extension field (GF(2^8) in this case)
set sizeOfField //size of the field we are building (65536 in this case)
set multiplier //multiplier in the shift register (aka constant term of the primitive polynomial used to create GF(2^16))
set newX to 0 //left box of the shift register
set oldX to 0 //temp storage for left box of the shift register
set newY to 1 //right box of the shift register
set oldY to 1 //temp storage for the right box of the shift register

write contents of shift register to outfile

for(n=1, n <= sizeOfField-2, n++) //create all elements of the new field except 0 and 1
{
 newX = oldX + oldY
 newY = oldX * multiplier
 use library to simplify newX and newY in the lower field (GF(2^8))
 set oldX to newX
 set oldY to newY
 write contents of shift register to outfile
 if the contents of the shift register are the element 1, then stop loop early
}



        close outfile
```

B. EXPONENTIAL ALGORITHM FOR GALOIS SHIFT REGISTER

The Mathematica program must be told which degree 2 polynomials are used to build the fields previous to the current field and does not use the previous shift register results. This takes a lot of processing time and requires a considerable amount of input from the user. However, there is another algorithm that uses Galois shift registers to build extension fields—we call the Exponential Algorithm [9]. This algorithm does not need computational algebra software and can be programmed in C++.

1. Exponential Algorithm Overview

Let α be a primitive element in GF(2^n). Suppose f(x) = $x^2 + x + \alpha^j$ is a primitive polynomial over GF(2^n). Then, use $f(x)$ to generate the Galois shift register. In the Standard Algorithm, the contents of the shift register will be elements of GF(2^n), and

therefore can be represented as either α^i for some i or as 0. In the Exponential Algorithm, the elements of the shift register will still represent α^i or 0, but will be represented by just the exponent i or *, respectively. (0 cannot represent the number zero because 0 in this case represents $\alpha^0 = 1$.) Also in the Exponential Algorithm, the operations of the shift register are modified but the result remains the same. For example, instead of new $s_0 = \alpha^j \bullet$ old s_1, we have new $s_0 = j +$ old s_1 (where the contents of s_0 and s_1 denote some exponent of α^i). This follow since multiplication of two numbers with the same base is accomplished by simply adding exponents. Also, since $0 \bullet \alpha^i = 0$, $* + i$ is defined to be *. Figuring out new s_1 is a little trickier. In the Standard Algorithm, the equation is new $s_1 =$ old $s_1 +$ old $s_0 \pmod 2$. However, in the Exponential Algorithm, new $s_1 =$ old $s_1 \oplus$ old s_0 where $\oplus$ is a new operator and is related to the addition (mod 2) operation from the Standard Algorithm. By definition, $* \oplus i$ is equal to i for any $0 \le i \le 2^n - 2$ since i represents powers of the primitive element α in GF(2^n). Intuitively, this has to do with the fact that $0 + \alpha^i = \alpha^i$. Along these lines, $i \oplus i = *$ since $\alpha^i + \alpha^i = 0 \pmod 2$. And $* \oplus * = *$ since $0 + 0 = 0$. However, for i and k not equal to each other and for neither i nor k equal to *, determining $i \oplus k$ requires information from the previously defined finite field. Note that if α is a primitive element of GF(2^n), then for α^i in GF(2^n) = $\{0, \alpha^0, \alpha^1, ..., \alpha^{2^n - 2}\}$, it must be the case that $0 \le i \le 2^n - 2$. So, i is an element of the additive group $\mathbb{Z}_{2^n - 1}$.

2. Exponential Algorithm—the $\oplus$ Operator

In more formal terms, to determine $s \oplus t$ when using a shift register to build the nonzero elements of GF(2^{2n}):

1. If $s = t$, the result is *.
2. If s = *, the result is t.
3. If t = *, the result is s.
4. At this point, both s and t are in $\mathbb{Z}_{2^n - 1}$, $s \ne t$, and neither s nor t are equal to *.
    i. Retrieve the 2 rows that represent α^s and α^t in the Galois shift register table that has the representations of all nonzero elements of GF(2^n).

ii. Do the $\oplus$ operation component-wise on the 2 rows (i.e., the polynomial representations of α^s and α^t). (In other words, return to Step 1 for each pair.)

iii. This results in a new row that represents α^u for some u in $\mathbb{Z}_{2^n-1}$. Return this result u.

3. Exponential Algorithm—An Example

Some rules of the operations + and $\oplus$ to remember are:

$$*\oplus i = i \qquad i \oplus i = * \qquad * + i = *$$

Also, the rules of the shift register for the Exponential Algorithm are:

$$\text{new } s_0 = j + \text{ old } s_1$$

$$\text{new } s_1 = \text{old } s_1 \oplus \text{old } s_0 \text{ (modulo } 2^n\text{)}$$

when the field being built is GF(2^{2n}).

For example, use the primitive polynomial $f(x) = x^2 + x + 1$ over GF(2) to build GF(2^2). Let a be a root of $f(x)$ and an element of GF(2^2). The Galois shift register generated by $f(x)$ in the Standard Algorithm is:

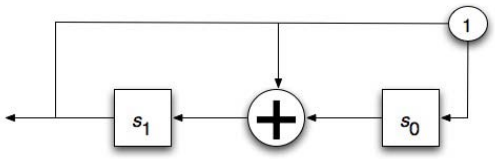


Figure 6. Galois Shift Register for Standard Algorithm Generated by $f(x)$

But in the Exponential Algorithm, the register is:

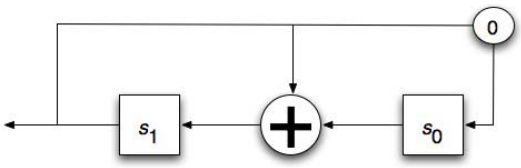


Figure 7. Galois Shift Register for Exponential Algorithm Generated by $f(x)$

(Recall that $1 = a^0$.) Initialize the register created for the Exponential Algorithm with a^0, which is denoted by * 0 in the shift register table. (Recall that $a^0 = 0 \bullet a + a^0 \bullet 1$. And the number 0 is denoted by * and a^0 is denoted by 0.)

Then, after the first step of the register,

$$\text{new } s_0 = 0 + \text{old } s_1 = 0 + * = *$$
$$\text{new } s_1 = \text{ old } s_1 \oplus \text{ old } s_0 = * \oplus 0 = 0.$$

After the second step,

$$\text{new } s_0 = 0 + \text{old } s_1 \pmod{2^1} = 0 + 0 = 0$$
$$\text{new } s_1 = \text{ old } s_1 \oplus \text{ old } s_0 = 0 \oplus * = 0.$$

After the third step,

$$\text{new } s_0 = 0 + \text{old } s_1 \pmod{2^1} = 0 + 0 = 0$$
$$\text{new } s_1 = \text{ old } s_1 \oplus \text{ old } s_0 = 0 \oplus 0 = *.$$

The resulting Galois shift register table for GF(2^2) is:

	a	1
a^0	*	0
a^1	0	*
a^2	0	0

Table 8. Nonzero Elements of GF(2^2) Created with Galois Shift Register and Exponential Algorithm

Next, consider the polynomial $g(x) = x^2 + x + a$ over GF(2^2). We saw $g(x)$ is primitive and that we can use it to build GF(2^4). Let b be a root of $g(x)$ and an element of GF(2^4). The shift register that $g(x)$ generates for the Exponential Algorithm is:

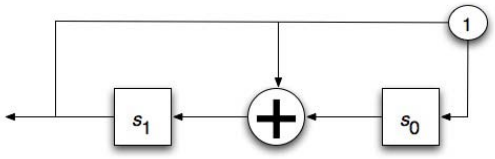


Figure 8. Galois Shift Register for Exponential Algorithm Generated by $g(x)$

After initializing the shift register with * 0, the first 3 rows of the shift register table are:

	b	1
b^0	*	0
b^1	0	*
b^2	0	1
b^3		
⋮		

Table 9. First Three Rows of Galois Shift Register Table for GF(2^4)

Then after the third step of the shift register,

$$\text{new } s_0 = 1 + \text{old } s_1 \pmod{2^2} = 1 + 0 = 1$$
$$\text{new } s_1 = \text{ old } s_1 \oplus \text{ old } s_0 = 0 \oplus 1 = ?$$

In order to determine the result of $0 \oplus 1$, we need to retrieve the rows of the previous shift register table that represent a^0 and a^1. These are * 0 and 0 *, respectively.

Next, we do the $\oplus$ operation on the rows, component-wise.

the first component of the row representing a^0

$\oplus$ the first component of the row representing a^1

is $*\oplus 0 = 0$. This will be the first component of the row we look for after doing the $\oplus$ operation on the second components of the above rows.

the second component of the row representing a^0

$\oplus$ the second component of the row representing a^1

is $0 \oplus * = 0$.

Thus, we look for the row whose components are 0 0 in the shift register table for GF(2^2). The row representing a^2 has components 0 0. So, return the result 2. Therefore, after the third step of the shift register generated by $g(x)$, we have 2 1.

The shift register continues to be stepped until the shift register table is complete.

The Exponential Algorithm can still be used with irreducible polynomials that are not primitive. However, the shift register then will only create a portion of the set of nonzero elements of the extension field.

	b	1
b^0	*	0
b^1	0	*
b^2	0	1
b^3	2	1
b^4	0	0
b^5	*	1
b^6	1	*
b^7	1	2
b^8	0	2

	b	1
b^9	1	1
b^{10}	*	2
b^{11}	2	*
b^{12}	2	0
b^{13}	1	0
b^{14}	2	2

Table 10. Nonzero Elements of GF(2^4) created with Galois Shift Register and Exponential Algorithm

C. C++ PROGRAM INSIGHTS

Unlike the Mathematica program, the C++ program uses the Exponential Algorithm to build fields using Galois shift registers. So, the C++ program does not need to do computations in the fields and runs much faster. For example, data was gathered on these extension fields and the positions of previous primitive roots within those fields with the Mathematica program over the course of 6 months. With the C++ program, we are able to gather about 8 times as much data in less than 4 minutes. However, the C++ program requires information from the previous extension fields, and uses more memory than the Mathematica program.

Another difference between the C++ program and the Mathematica program is the way irreducible polynomials of the form $x^2 + x + \alpha^j$ are found. In the C++ program, we use the trace of the element α^j to find irreducible polynomials of the form $x^2 + x + \alpha^j$ instead of the method used in the Mathematica program. The trace of an element and this method is explained in the section below.

For the Mathematica program, the only way to determine if the irreducible polynomial is primitive is to see if the shift register generated by it has the maximal period. If it does not, then the polynomial is not primitive. The C++ program uses Theorem 3 from the Results Chapter to quickly determine if an irreducible polynomial of

the form $x^2 + x + \alpha^j$ is primitive. If it is primitive, then the shift register generated by the polynomial is run using the Exponential Algorithm.

The output of the C++ program is a text file listing the degree 2 polynomials used to build the fields up to a particular field as well as the power of the primitive root (of the current field) that is equal to the primitive roots used to generate the previous extension fields. An example is in Appendix C. Once the memory problem becomes too great to build a particular field GF(2^{2n}), the program is still able to print out which polynomials of the form $x^2 + x + \alpha^j$ are primitive over GF(2^n) since only information about fields GF(2^n) and smaller are needed to determine that.

1. Trace

We can use the trace of an element α^j of a field GF(2^n) to determine if the polynomial $x^2 + x + \alpha^j$ is irreducible over that field. The *trace* is defined to be the sum of the conjugates of an element of a field. In the Exponential Algorithm for Galois shift registers, we define the trace specifically as the $\oplus$ operation performed on all of the shift register table representations of the conjugates of α^j [9]. If the trace is * * (e.g., the equivalent of the number "zero" in the field), then the polynomial $x^2 + x + \alpha^j$ is reducible. However, if the result is * 0 (e.g., the equivalent of the number "one" in the field), then the polynomial $x^2 + x + \alpha^j$ is irreducible.

For example, consider the polynomial $x^2 + x + b$ over the field GF(2^4) where b is a primitive element of GF(2^4). We want to determine if the polynomial is irreducible over this field. The trace of b^1 is $b^1 + b^2 + b^4 + b^8$. So, we retrieve the corresponding rows from Table 3 above and get the equation:

$$\begin{array}{r} 0\ * \\ 0\ 1 \\ 0\ 0 \\ \oplus\ 0\ 2 \\ \hline * \ * \end{array}$$

The result is * *, so the polynomial is reducible over GF(2^4).

Now consider the polynomial $x^2 + x + b^7$ over GF(2^4). The trace of b^7 is $b^7 + b^{14} + b^{11} + b^{13}$. Retrieving the appropriate rows from Table 3, we get the equation:

$$\begin{array}{r} 1\ 2 \\ 2\ 2 \\ 1\ 0 \\ \oplus\ 2\ * \\ \hline * \ 0 \end{array}$$

The result is * 0, so the polynomial is irreducible over GF(2^4).

2. C++ Program Pseudocode

The pseudocode for the C++ Program that iteratively builds the fields GF(2^4) up to GF(2^{32}) is below and the actual code is in Appendix D.

Define a galois_table structure that will hold info about each extension field built. This structure will include the Galois shift register table.
Define the variable STAR to be the largest unsigned integer the cpu can handle.
Initialize the register (which is just 2 integers) to STAR 0.
Start with a primitive polynomial of the form $x^2 + x + \alpha^j$ over the field GF(2^4) (since we know the only two choices).
Loop until GF(2^{32}):
{
    Loop until out of primitive polynomials for the field:
    {
        Build the shift register table for the next extension field using the Exponential Algorithm and the next primitive polynomial in the list.
        When you see the previous primitive root during this procedure (looks like 0 STAR in the shift register table), note the power of the current primitive root it is equal to.
        Determine which polynomials of the form $x^2 + x + \alpha^j$ are irreducible over the current field by using the trace of the constant coefficient.
        Use Theorem 2 from the Results Chapter to make a list of which of those irreducible polynomials are primitive.
    }
}

IV. RESULTS

The results of this work include two programs, charts detailing the extension fields and primitive roots, as well as some insight into the field that AES S-box computations are computed over.

A. CHARTS

The charts in Appendix E are a visualization of the location of previous primitive roots in extension fields from GF(2^2) to GF(2^{16}). Each box represents a separate extension field. Within each box is listed the polynomial used to create that extension and the power of the root of that polynomial which is equal to each root of the polynomials used to create each of the previous extension fields.

The polynomial $f(x) = x^2 + x + 1$ is irreducible over GF(2). Let a be a root of that polynomial. Use the polynomial to create the field GF(2^2) = $\{0, 1, a, a+1\}$. As shown in Chapter II, $x^2 + x + a$ is irreducible over GF(2^2). Let b be the root of $x^2 + x + a$. We create the field GF(2^4) with the primitive polynomial $x^2 + x + a$ over GF(2^2). In this case, a is our only previous primitive root, so we note which power of b is equal to the root a in the box ($b^5 = a$).

Note that the vertical lines extending from box to box designate which fields the extension fields are built upon. Also, boxes that use the same color indicate that the constant coefficients of the primitive polynomials used to build those fields are in the same conjugacy class. The field extensions that the primitive polynomials are used to create are indicated in the left margin.

B. THEOREMS

We restrict the choice of polynomial to build the extension field to be of the form $x^2 + x + \alpha^i$ or $x^2 + \alpha^i x + 1$, as indicated in Chapter I. The AES polynomial is $x^8 + x^4 + x^3 + x + 1$, and the field it creates cannot be realized as an extension field if we only use polynomials of the above forms.

Theorem 1: The field that the AES S-Boxes are implemented in cannot be built with degree two extensions of the form $x^2 + x + \alpha^i$ or $x^2 + \alpha^i x + 1$.

Proof:

The AES polynomial, $x^8 + x^4 + x^3 + x + 1$, is not primitive, of period 51. The only polynomial which is primitive of degree 2 over GF(2) to create GF(2^2) is $x^2 + x + 1$, as we showed in Chapter II. Then the only polynomial we can use to create GF(2^4) of our form is $x^2 + x + a$ where a is a root of $x^2 + x + 1$. (Any other choice of polynomial is isomorphic to this choice.) To create a field of 2^8 elements we have a choice of $x^2 + x + b^3$ or $x^2 + x + b^7$, where b is a root of $x^2 + x + a$. However, $x^2 + x + b^3$ is irreducible but of period 85. (This can be shown with Conway's method [2].) So, it is not primitive. The conjugates of b^3 — b^6, b^{12}, b^9 — yield the same results, as we discussed in Chapter II. Choosing $x^2 + x + b^7$ results in a primitive polynomial of period 255 as do the conjugates of b^7 — b^{14}, b^{13}, b^{11}. All other choices of the polynomial $x^2 + x + b^i$ are reducible for i = 0, 1, 2, 4, 5, 8, and 10.

Next, consider using polynomials of the form $x^2 + b^i x + 1$ to build the field GF(2^8) from GF(2^4) where b is an element of GF(2^4). The polynomial is only irreducible when $i = 1$ or $i = 3$. In each case, the polynomial is not primitive, with period 17. Therefore, the degree 2 polynomial that builds the exact same field as the AES polynomial must be of the form $x^2 + b^i x + b^j$ with neither $b^i = 1$ nor $b^j = 1$. □

Incidentally, one can use Magma to show that $x^8 + x^4 + x^3 + x + 1$ builds the same exact field—GF(2^8) —from GF(2) as $x^2 + bx + b^5$ will build from GF(2^4). An example of the Magma commands [10] used to show this is below.

```
k := GF(256);
P<x> := PolynomialRing(k);
a := Roots(x^2+x+1)[1][1];
b := Roots(x^2+x+a)[1][1];
c := Roots(x^2+b*x+b^5)[1][1];
aes := Roots(x^8+x^4+x^3+x+1);
c in [r[1] : r in aes};
S51 := [i: i in k | i^51 eq 1];
```

Note further that $x^8 + x^6 + x^5 + x^3 + 1$ builds the same field as $x^2 + x + b^7$.

One of the first things we want to determine is if a polynomial of the form $x^2 + x + \alpha^i$ is primitive over a particular field. It turns out that in creating the field with the Galois shift register using the exponent algorithm, a pattern emerges from the elements of the table. This pattern is directly related to the coefficients of the polynomial. We can use this pattern to determine the period of the polynomial without running the whole shift register. In fact, if a polynomial is of the form $x^2 + \alpha^i x + \alpha^j$, we do not need to run the shift register at all.

For example, we use $x^2 + x + 1$ to build GF(2^2). Let *a* be a root of $x^2 + x + 1$. Then $a^2 + a + 1 = 0$. Recall that in the table of nonzero field elements, * refers to the number zero and the integers refer to the power of 1 (a primitive element in the field GF(2^{2-1}) = GF(2)). For example, 0 in the table represents $1^0 = 1$. Also, the rows represent linear combinations of *a* and 1. For instance, * 0 in the a^0 row represents $0 \bullet a + 1^0 \bullet 1 = 1$. This makes sense because $a^0 = 1$. For our table of nonzero field elements, we then get:

	a	1
a^0	*	0
a^1	0	*
a^2	0	0

Table 11. Nonzero Elements of GF(2^2) created with Galois Shift Register

The tables of nonzero field elements created with Galois Shift registers are explained in Chapter II.

Now, use $x^2 + x + a$ to build GF(2^4) as an extension of GF(2^2). Let *b* be a root of $x^2 + x + a$. Then $b^2 + b + a = 0$. Since *a* is the primitive element in the field that we built GF(2^4) from, the integers in the table refer to powers of *a*. For example, 0 in the table really represents $a^0 = 1$, 1 represents $a^1 = a$, and 2 represents $a^2 (= a + 1)$. As in the

other tables, * refers to the number zero. Also, the rows represent linear combinations of b and 1. For instance, * 0 in the b^0 row represents $0 \bullet b + a^0 \bullet 1 = 1$. This makes sense because $b^0 = 1$. The table of nonzero field elements looks like:

	b	1
b^0	*	0
b^1	0	*
b^2	0	1
b^3	2	1
b^4	0	0
b^5	*	1
b^6	1	*
b^7	1	2
b^8	0	2
b^9	1	1
b^{10}	*	2
b^{11}	2	*
b^{12}	2	0
b^{13}	1	0
b^{14}	2	2

Table 12. Nonzero Elements of GF(2^4) created with Galois Shift Register

Note that a pattern can be observed across the rows when we cut the table into sections. For instance, to get from b^0 to b^5 to b^{10}, one only needs to add 1 1 to the entries in the table, i.e., we multiply by b^5 by adding the powers.

	b	1		b	1		b	1
b^0	*	0	b^5	*	1	b^{10}	*	2
b^1	0	*	b^6	1	*	b^{11}	2	*
b^2	0	1	b^7	1	2	b^{12}	2	0
b^3	2	1	b^8	0	2	b^{13}	1	0
b^4	0	0	b^9	1	1	b^{14}	2	2

Table 13. Rearranged Nonzero Elements of GF(2^4) Created with Galois Shift Register

In general, we are building extension fields using degree 2 polynomials, i.e., quadratic extensions. If we let β be one root of the irreducible polynomial $f(x) = x^2 + \alpha^i x + \alpha^j$ over GF(2^n), we know there is only one other root. Call this other root β^k. Then, $f(x) = (x+\beta)(x+\beta^k) = x^2 + (\beta + \beta^k)x + \beta^{k+1}$. Therefore, $\beta^{k+1} = \alpha^j$ and $\beta + \beta^k = \alpha^i$. If we assume we are building the fields using the exponential algorithm explained in Chapter III, this means that the row representing β^k will look like 0 0.

Interestingly, it turns out the root β^k is actually β^{2^n} as shown in the following lemma.

Lemma 1: Let $f(x) = x^2 + ax + c$ be irreducible over GF(2^n). Let β be a root. Then the other root of $f(x)$ is β^{2^n} [3].

Proof:

$$\left[f(x)\right]^{2^n} = \left[x^2 + ax + c\right]^{2^n}$$

$$= \left(x^2\right)^{2^n} + \left(ax\right)^{2^n} + c^{2^n} \text{ since all cross terms are even and are } \equiv 0 \bmod 2$$

$$= x^{2\cdot 2^n} + a^{2^n} x^{2^n} + c^{2^n}$$

$$= x^{2\cdot 2^n} + ax^{2^n} + c \text{ since all the elements of the field GF}(2^n) \text{ satisfy } b^{2^n} = b$$

$$= x^{2^{n+1}} + ax^{2^n} + c = \left(x^{2^n}\right)^2 + a\left(x^{2^n}\right) + c = f\left(x^{2^n}\right)$$

Since $f(\beta)=0$, we get $f\left(\beta^{2^n}\right)=0$. So, β^{2^n} is also a root of *f*(*x*). □

Note that $\beta^{2^{2n}}=\beta$ in GF(2^{2n}), so β^{2^n} is distinct from β in GF(2^{2n}) if $n>0$.

Therefore, if we let $f(x)=x^2+\alpha^i x+\alpha^j$ be an irreducible polynomial over GF(2^n) and β be a root, then β^{2^n} is also a root of *f*(*x*). Note that we will use *f*(*x*) to build GF(2^{2n}) if this polynomial is also primitive.

Theorem 2: The order of an irreducible polynomial $f(x)=x^2+\alpha^i x+\alpha^j$ over GF(2^n) is equal to the order of the element *j* in the additive group $\mathbb{Z}_{2^n-1}$.

Proof:

Recall from our comments above that $\alpha^j=\beta^{2^n+1}$. So, β^{2^n+1} is represented by * *j* in the shift register table. Since $\beta^{2^n+1}=\alpha^j$, $(\beta^{2^n+1})^2=(\alpha^j)^2=\alpha^{2j}$. So, the entry for $\beta^{(2^n+1)2}$ is * 2*j* in the table. Similarly, the entry for $\beta^{(2^n+1)k}$ is * *kj* (mod 2^n – 1 because these numbers represent powers of a primitive element in GF(2^n)). □

Theorem 3: If $f(x)=x^2+\alpha^i x+\alpha^j$ is irreducible over GF(2^n) and β in GF(2^{2n}) is a root, then the other root, β^{2^n}, is represented as 0 *i* in the Galois shift register table

Proof:

Since β and β^{2^n} are roots of *f*(*x*), *f*(*x*) = $(x+\beta)(x+\beta^{2^n})$ which equals $x^2+(\beta+\beta^{2^n})x+\beta^{2^n+1}$. So, $\alpha^i=\beta+\beta^{2^n}$ and $\alpha^j=\beta^{2^n+1}$. Since $\alpha^i=\beta+\beta^{2^n}$, then $\beta^{2^n}=\alpha^i+\beta$. We know β is represented by 0 * in the Galois shift register table when using the exponent algorithm. So,

$$\begin{array}{rl} \beta+\alpha^i = & 0\ * \\ \oplus & \underline{*\ \ i} \\ & 0\ \ i \end{array}.$$

Therefore, β^{2^n} is represented as 0 *i* in the Galois shift register table. □

Using the second program described in Chapter III, it is simple to determine which polynomials of the form $x^2 + x + \alpha^i$ are primitive over GF(2^{32}) and can be used to build GF(2^{64}).

C. CONCLUSIONS

The results of this work are the first steps towards a full understanding of the field that AES computation occurs in—GF(2^8). The charts created with the data from the C++ program detail which power of the current primitive root is equal to previous primitive roots for fields up through GF(2^{16}) created by polynomials of the form $x^2 + x + \alpha^i$ for a primitive element α. Currently, the C++ program will also provide all the primitive polynomials of the form $x^2 + x + \alpha^i$ for a primitive element α over the fields through GF(2^{32}). This work also led to a deeper understanding of certain elements of a field and their equivalent shift register state when using the Exponential Algorithm. In addition, given an irreducible polynomial $f(x) = x^2 + \alpha^i x + \alpha^j$ over GF(2^n), the period (and therefore the primitivity) can be determined without running the shift register generated by $f(x)$.

THIS PAGE INTENTIONALLY LEFT BLANK

V. FUTURE WORK

There are still unanswered questions left to explore when it comes to understanding the field—GF(2^8)—that AES relies on.

A. OTHER ALGORITHMS

While being able to build the fields with shift registers and program this method saved a lot of time, we still ran into some stumbling blocks. The Mathematica program using the Standard Algorithm did not need to store much in memory, but it took a long time to do its computations. On the other hand, the C++ program using the Exponential Algorithm needed a lot of memory but very little run time. Perhaps there is a different algorithm that is more in the middle of the resource spectrum—one that can build these fields quickly with shift registers but does not require as much memory as the Exponential Algorithm. Or maybe there is a better way to design the C++ program while still using the Exponential Algorithm.

B. AES AND POLYNOMIALS OF THE FORM $x^2 + x + \alpha^i$

In his paper [1], Canright explored building extensions fields with polynomials of the form $x^2 + \alpha x + \beta$ over GF(2^n) where α and β are elements of GF(2^n) and where one of the α or β (but not both) are equal to 1. With polynomials of this form, he is able to create an implementation of an S-box that is 16% smaller than the previous most efficient implementation. By modifying the implementation of AES using polynomials of the form $x^2 + x + \alpha^i$ where α is a primitive element, can an implementation that is more efficient than Canright's be found?

In addition to determining if using polynomials of the form $x^2 + x + \alpha^i$ to build GF(2^8) makes the AES implementation more efficient, there are also other questions regarding polynomials of the form $x^2 + x + \alpha^i$ and AES. For example, would using polynomials of this form to implement AES have the adverse effect of weakening the AES algorithm in some way?

The relationship between the roots of these polynomials, the constant coefficients of these polynomials, and the AES S-boxes needs more investigation.

C. MATHEMATICS AND POLYNOMIALS OF THE FORM $x^2 + x + \alpha^i$

Besides asking questions regarding these polynomials and AES, there are also interesting mathematical questions. Specifically, is there a relationship among the powers of the primitive roots used to generate the coefficients of each polynomial $x^2 + x + \alpha^i$ that are, in turn, used to build the field extensions? Can we predict what polynomials will be primitive? Also, can one continue the field extensions forever using only polynomials of the form $x^2 + x + \alpha^i$ for some primitive element α? An argument can be made using counting ideas to indicate that this is probably possible. It would be very nice to be able to predict their form.

APPENDIX A. ONE PAGE OF MATHEMATICA PROGRAM OUTPUT EXAMPLE

Appendix A is a one-page example of the output generated by the Mathematica program in Appendix B. The output shows the power of the root of the polynomial that generates the shift register followed by the contents of the storage elements of the shift register separated by a comma.

```
D0  0, 1
D1  1, 0
D2  1, 1 + a + b + (1 + a + a b) c
D3  a + b + (1 + a + a b) c, 1 + a + b + (1 + a + a b) c
D4  1, 1 + a + a b + b c
D5  a + a b + b c, 1 + a + b + (1 + a + a b) c
D6  1 + (1 + a) b + (1 + a + (1 + a) b) c, 1 + a + b + (1 + a b) c
D7  a + a b + (a + b) c, 1 + a + a b + (a + b) c
D8  1, a + (1 + (1 + a) b) c
D9  1 + a + (1 + (1 + a) b) c, 1 + a + b + (1 + a + a b) c
D10  b + (a + b) c, a + c
D11  a + b + (1 + a + b) c, a + (1 + a) b + (a + (1 + a) b) c
D12  a b + (1 + a b) c, a b + (1 + a + (1 + a) b) c
D13  (a + b) c, a + (1 + a) c
D14  a + (1 + b) c, b + (1 + a b) c
D15  a + b + (1 + a) b c, 1 + a + b + (1 + a) b c
D16  1, c
D17  1 + c, 1 + a + b + (1 + a + a b) c
D18  a + b + (a + a b) c, a b + (1 + a + b) c
D19  a + (1 + a) b + (1 + (1 + a) b) c, b + a b c
D20  a + a b + (1 + b) c, b c
D21  a + a b + c, a b + (1 + b) c
D22  a + b c, 1 + a b + a b c
D23  1 + a + a b + (1 + a) b c, a b
D24  1 + a + (1 + a) b c, a + b c
D25  1 + a b c, 1 + (1 + a) b + (1 + (1 + a) b) c
D26  (1 + a) b + (1 + b) c, 1 + b + (1 + (1 + a) b) c
D27  1 + a b + a b c, 1 + a + a b + (1 + a + (1 + a) b) c
D28  a + (1 + a + b) c, a + a b + b c
D29  a b + (1 + a) c, a + a b c
D30  a + a b + (1 + a + a b) c, 1 + b + c
D31  1 + a + (1 + a) b + (a + a b) c, a + (1 + a) b + (a + a b) c
D32  1, (1 + a) b + (a + a b) c
D33  1 + (1 + a) b + (a + a b) c, 1 + a + b + (1 + a + a b) c
D34  a + a b + c, 1 + b + (1 + a + b) c
D35  1 + a + (1 + a) b + (a + b) c, 1 + a b + a b c
D36  a + b + (a + (1 + a) b) c, 1 + a + (1 + a) b + c
D37  1 + a b + (1 + a + (1 + a) b) c, 1 + b + (1 + b) c
D38  (1 + a) b + (a + a b) c, 1 + c
D39  1 + (1 + a) b + (1 + a + a b) c, a + (1 + a) b c
D40  1 + a + (1 + a) b + (1 + a + b) c, a + a b + (1 + a + a b) c
D41  1 + b + (1 + a) b c, (1 + (1 + a) b) c
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. MATHEMATICA PROGRAM CODE

Appendix B is the Mathematica code for creating the elements of GF(2^{16}) using a Galois shift register generated by $x^2 + x + c^{21}$ for c in GF(2^8).

```
SetDirectory["C:\\Documents and Settings\\jody\\Desktop\\thesis"];
<<AlgFields.txt

(*ClearAll[fieldTable, irredTable];*)

FDeclareFiniteField[GF2,2];
FDeclareExtensionField[GF4,GF2,{a^2+a+1}];
FDeclareExtensionField[GF16, GF4, {b^2+b+a}];
FDeclareExtensionField[GF256, GF16, {c^2+c+b^7}];
FDeclareExtensionField[GF65536, GF256, {d^2+d+c^21}];

(*things you need to change each time*)
outFile = OpenWrite["GF65535poly21.txt"];
xtnField = GF65536;
lowerField = GF256;
sizeOfField = 65536;
multiplier = c^21;
newTerm = d;

newX =0;
oldX =0;
newY =1;
oldY =1;


Print[newTerm, "0  ",newX,", ", newY];
WriteString[outFile,"D0  ",newX,", ", newY, "\n"];

For[n=1, n≤sizeOfField-2, n++,
  newX=oldX+oldY;
  newY = oldX * multiplier;
  newX = FSimplifyE[newX, lowerField];
  newY = FSimplifyE[newY, lowerField];
  oldX=newX;
  oldY=newY;
  (*Print[newTerm, n,"  ",newX,", ",newY];*)
```

```
  WriteString[outFile, "D", n,"  ",newX,", ",newY, "\n"];
  ]
Close[outFile];
```

APPENDIX C. ONE PAGE OF C++ PROGRAM OUTPUT EXAMPLE

Appendix C is a one-page example of the output generated by the C++ program in Appendix D. The output states the primitive polynomials used to build each field up to that point and also which power of the current primitive root is equal to previous primitive roots. For example, consider the polynomial $x^2 + x + a$ over GF(2^2) where a is an element of GF(2^2). Suppose b is a root of the polynomial. Then, in GF(2^4), $b^5 = a$. In the output of the program, this is worded as "position of root from degree 4 extension is: 5".

```
STAR is ffffffff
Building GF4..

building GF16 with x^2+x+a^1
position of root from degree 4 extension is: 5

GF4 built with x^2+x+1. GF16 built with x^2+x+a^1. GF256 built with
x^2+x+b^7
position of root from degree 8 extension is: 221
position of root from degree 4 extension is: 85

GF4 built with x^2+x+1. GF16 built with x^2+x+a^1. GF256 built with
x^2+x+b^7. GF2to16 built with x^2+x+c^11.
position of root from degree 16 extension is: 29812
position of root from degree 8 extension is: 34952
position of root from degree 4 extension is: 43690

GF4 built with x^2+x+1. GF16 built with x^2+x+a^1. GF256 built with
x^2+x+b^7. GF2to16 built with x^2+x+c^22.
position of root from degree 16 extension is: 14906
position of root from degree 8 extension is: 17476
position of root from degree 4 extension is: 21845

GF4 built with x^2+x+1. GF16 built with x^2+x+a^1. GF256 built with
x^2+x+b^7. GF2to16 built with x^2+x+c^44.
position of root from degree 16 extension is: 7453
position of root from degree 8 extension is: 8738
position of root from degree 4 extension is: 43690

GF4 built with x^2+x+1. GF16 built with x^2+x+a^1. GF256 built with
x^2+x+b^7. GF2to16 built with x^2+x+c^88.
position of root from degree 16 extension is: 36494
position of root from degree 8 extension is: 4369
position of root from degree 4 extension is: 21845
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. C++ PROGRAM CODE

Appendix C contains the C++ program that builds fields using Galois shift registers with the Exponential Algorithm. It also is capable of finding primitive polynomials of the form $x^2 + x + \alpha^i$ for a primitive element α over these fields.

```
/*Jody Radowicz
Masters Thesis, 2006
This program iteratively goes through all of the extension fields over
GF2 through GF(2^16) and prints out the primitive polynomials used to
build the extensions as well as which power of the current root is
equal to each previous root. It also reports which polynomials are
reducible as well as the irreducible but imprimitive polynomials with
their periods.
***Note***: This program currently only ever considers polynomials of
the form x^2+x+constant
*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <limits.h>
#include <math.h>
#include <vector>
using namespace std;


typedef u_int32_t int_type; //a definition of the largest int,
depending on the computer
typedef vector<int_type> NumVector; //holds the constant coefficients
of the primitive polynomials of the form x^2+x+root^i over the current
field

typedef struct galois_table_struct //struct that holds information
about a particular field
{
      int_type **curr;//pointer to a pointer to the current field
      struct galois_table_struct *prev;//pointer to the struct that
holds info about the field that the current field is extended from
      struct galois_table_struct *next;//pointer to the struct that
holds info about the extension field built from this field
      int field_size; //size of table that holds the possible contents
of the shift register + 1, also happens to be the size of the field
      int extn_degree;//degree of the extension field over GF2
      int prev_field_size;//size of previous extension field
      int_type root_position;//position in the table created with the
Galois shift register of the root of the primitive poly used to create
previous field (means new_root^root_position = old_root)
      char root_name;//used to keep track of root name for printing
purposes
}     galois_table;
```

```
int_type STAR = UINT_MAX;//STAR is a constant int that is treated differently when it comes to multiplication and addition. It actually represents the number zero while the other numbers all represent the exponent i on the number root^i when it comes to the contents of the shift registers

int_type curr_left_reg =STAR;//the left register of the Galois shift register
int_type old_left_reg = STAR;//a temp holder for the left register
int_type curr_rt_reg = 0;//the right register of the Galoid shift register
int_type old_rt_reg = 0;//a temp holder for the right register

//declare one instance for each type of table since we will only need one table for each extension at a time
galois_table GF4_table;
galois_table GF16_table;
galois_table GF256_table;
galois_table GF2to16_table;
//galois_table GF2to32_table;

int_type sanity_check_num = 0; //number of bits of machine - 1, meant to avoid overflow when adding large numbers

void print_table (galois_table t); //prints the table created from the Galois register, which holds the elements of the extension field whose galois_table struct gets passed to it
void build_extn_field(galois_table &t, int_type multiplier);//builds the extension field, using a polynomial of the form x^2+x+root^multiplier
void build_table_memory();//builds table memory
NumVector coset_trace(galois_table table);//determines the trace of the constant coefficients of polynomials and returns a vector of primitive polynomials' constant coefficients with which to build the next extension field. The trace s used to determine if hte polynomial is reducible or not.
int_type check_order(int_type number, galois_table table); //returns the order of number in the given Galois field. When passed a constant coefficient, we can determine if the corresponding irreducible polynomial is primitive or not by checking the coefficient's order in the previous Galois field.
void calc_roots(galois_table table, int_type top_field_size, int_type prev_root_pos, int_type times);//prints where all of the previous roots occur in a given field. top_field_size the is the size of the highest extension field, times is the degree of the highest extension field over GF2 and helps the function determine how many roots it needs to look for.
int_type high_bit_pos(int_type number); //returns the number of the highest bit position (start counting with 0)

//struct //used for command line arguments
//{
//    int_type size_of_field;
//} global_cfg;

void usage() //prints the form of the command used to run the program
{
```

```
      printf("./iter_fields \n");
}

void parse_args(int argc, char **argv) //parses command line arguments,
if there are any
{
      if (argc != 1)
            usage();
//    else
//          global_cfg.size_of_field = atoi(argv[1]);

      //--Sanity checking arguments
//    if (global_cfg.size_of_field != 16)
//    {
//          printf("parse_args::Error! Size should be 16.\n");
//          exit(0);
//    }

//    printf("Doing      run      with      field      size:      %d\n",
global_cfg.size_of_field);

}
void  dump_regs(galois_table  table,  int  row)  //puts  the  register
contents into a particular field's table. Used when building the tables
that hold the elements of a particular field.
{
      table.curr[row][0] = curr_left_reg;
      table.curr[row][1] = curr_rt_reg;
}

void reset_regs()//resets the registers to the starting state
{
      curr_left_reg = old_left_reg = STAR;
      curr_rt_reg = old_rt_reg = 0;
}


void  build_table_memory()//builds  tables,  makes  sure  there  is  enough
memory.  Fills  in  some  of  the  galois_table  struct's  elements  that  are
common  to  a  particular  extension  degree.  Only  done  once  at  the
beginning of the program.
{

GF4_table.curr     =    new int_type*[3];
GF16_table.curr    =    new int_type*[15];
GF256_table.curr   =    new int_type*[255];
GF2to16_table.curr =    new int_type*[65535];
//GF2to32_table.curr =  new int_type*[4294967295];


if ( (!GF4_table.curr) || (!GF16_table.curr) || (!GF256_table.curr) ||
(!GF2to16_table.curr) /*|| (!GF2to32_table.curr)*/ )
{
      fprintf(stderr,  "Error  allocating  initial  dimension  of  table
memory. exiting\n");
      exit(0);
}
```

```
for (int i = 0; i < 3; i++)
{
      GF4_table.curr[i] = new int_type[2];
      if (GF4_table.curr[i] == NULL)
      {
            printf("Error   allocating   sub-array   in   GF4.   i   =   %d
exiting\n", i);
            exit(0);
      }
}
      GF4_table.field_size = 4;
      GF4_table.extn_degree = 2;
      GF4_table.prev_field_size=1;
      GF4_table.prev = NULL;
      GF4_table.next = &GF16_table;
      GF4_table.root_name = 'a';

for (int i = 0; i < 15; i++)
{
      GF16_table.curr[i] = new int_type[2];
      if (GF16_table.curr[i] == NULL)
      {
            printf("Error   allocating   sub-array   in   GF16.   i   =   %d
exiting\n", i);
            exit(0);
      }
}
      GF16_table.field_size = 16;
      GF16_table.extn_degree = 4;
      GF16_table.prev_field_size = 4;
      GF16_table.prev = &GF4_table;
      GF16_table.next = &GF256_table;
      GF16_table.root_name = 'b';

for (int i = 0; i < 255; i++)
{
      GF256_table.curr[i] = new int_type[2];
      if (GF256_table.curr[i] == NULL)
      {
            printf("Error   allocating   sub-array   in   GF256.   i   =   %d
exiting\n", i);
            exit(0);
      }
}
      GF256_table.field_size = 256;
      GF256_table.extn_degree =8;
      GF256_table.prev_field_size = 16;
      GF256_table.prev = &GF16_table;
      GF256_table.next = &GF2to16_table;
      GF256_table.root_name = 'c';


for (int i = 0; i < 65535; i++)
{
      GF2to16_table.curr[i] = new int_type[2];
      if (GF2to16_table.curr[i] == NULL)
```

```
	{
		printf("Error allocating sub-array in GF2to16. i = %d
exiting\n", i);
		exit(0);
	}
}
	GF2to16_table.field_size = 65536;
	GF2to16_table.extn_degree = 16;
	GF2to16_table.prev_field_size = 256;
	GF2to16_table.prev = &GF256_table;
//	GF2to16_table.next = &GF2to32_table;
	GF2to16_table.next = NULL;
	GF2to16_table.root_name = 'd';

/*for (int i = 0; i < (pow(2, 32)-1); i++)
{
	GF2to32_table.curr[i] = new int_type[2];
	if (GF2to32_table.curr[i] ==NULL)
	{
		fprintf(stderr, "Error allocating sub-array in GF2to32. i =
%d exiting\n", i);
		exit(0);
	}
}

	GF2to32_table.field_size = (int) pow(2, 32);
	GF2to32_table.extn_degree = 32;
	GF2to32_table.prev_field_size = 65536;
	GF2to32_table.prev = &GF2to16_table;
	GF2to32_table.next = NULL;
	GF2to32_table.root_name = 'e';

*/
	//build GF4's table. It is the same every time because there is
only one primitive polynomial over GF2, x^2+x+1.
	printf("Building GF4..\n");
	GF4_table.curr[0][0] = STAR;
	GF4_table.curr[0][1] = 0;
	GF4_table.curr[1][0] = 0;
	GF4_table.curr[1][1] = STAR;
	GF4_table.curr[2][0] = 0;
	GF4_table.curr[2][1] = 0;


}

int main(int argc, char **argv)
{

	NumVector V;//vector that holds the constant coefficients of
primitive polynomials over the current field
	NumVector::iterator my_iter;//an iterator that runs through the
vector
	parse_args(argc, argv);

	sanity_check_num = high_bit_pos(STAR);//sanity_check_num is used
to make sure there are no overflows in the calculations
```

```
      printf("STAR is %x\n", STAR);
      build_table_memory();

//********do below when you want to follow one path down the field
extensions
      printf("\nbuilding   GF16   with   x^2+x+%c^1",   GF16_table.prev-
>root_name);
      build_extn_field(GF16_table, 1);
      calc_roots(GF16_table,          GF16_table.field_size,          1,
GF16_table.extn_degree);

      reset_regs();
      printf("\nGF4 built with x^2+x+1. GF16 built with x^2+x+%c^1.
GF256     built     with     x^2+x+%c^7",     GF16_table.prev->root_name,
GF256_table.prev->root_name);
      build_extn_field(GF256_table, 7);
      calc_roots(GF256_table,         GF256_table.field_size,         1,
GF256_table.extn_degree);

      reset_regs();
      printf("\nGF4 built with x^2+x+1. GF16 built with x^2+x+%c^1.
GF256  built  with  x^2+x+%c^7.  Gf2to16  build  with  x^2+x+%c^11.",
GF16_table.prev->root_name,                GF256_table.prev->root_name,
GF2to16_table.prev->root_name);
      build_extn_field(GF2to16_table, 11);
      calc_roots(GF2to16_table,       GF2to16_table.field_size,       1,
GF2to16_table.extn_degree);


/*    reset_regs();
      printf("\nGF4 built with x^2+x+1. GF16 built with x^2+x+%c^1.
GF256 built with x^2+x+%c^7. Gf2to16 build with x^2+x+%c^11. GF2to32
built with x^2+x+%c^19.", GF16_table.prev->root_name, GF256_table.prev-
>root_name,      GF2to16_table.prev->root_name,      GF2to32_table.prev-
>root_name);
      build_extn_field(GF2to32_table, 19);
      calc_roots(GF2to32_table,       GF2to32_table.field_size,       1,
GF2to32_table.extn_degree);
*/
//*******end straight run through fields

//*******do  below  when  you  want  to  run  through  all  of  the  fields
iterively
/*
      printf("\nbuilding   GF16   with   x^2+x+%c^2",   GF16_table.prev-
>root_name);
      build_extn_field(GF16_table, 2);
//    print_table(GF16_table);
      V = coset_trace(GF16_table);
      calc_roots(GF16_table,          GF16_table.field_size,          1,
GF16_table.extn_degree);

      my_iter = V.begin();
      while(my_iter != V.end())
      {
            reset_regs();
```

```
            printf("\nGF4 built with x^2+x+1. GF16 built with
x^2+x+%c^2. GF256 built with x^2+x+%c^%d", GF16_table.prev->root_name,
GF256_table.prev->root_name, *my_iter);
            build_extn_field(GF256_table, *my_iter);

            NumVector newV;
            NumVector::iterator new_iter;

            newV = coset_trace(GF256_table);

            calc_roots(GF256_table,      GF256_table.field_size,      1,
GF256_table.extn_degree);

            new_iter = newV.begin();
            while(new_iter != newV.end())
            {
                  reset_regs();
                  printf("\nGF4 built with x^2+x+1. GF16 built with
x^2+x+%c^2. GF256 built with x^2+x+%c^%d. GF2to16 built with
x^2+x+%c^%d.", GF16_table.prev->root_name, GF256_table.prev->root_name,
*my_iter, GF2to16_table.prev->root_name, *new_iter);
//                printf("\nGF2to16    built    with    x^2+x+root^%d\n",
*new_iter);
                  build_extn_field(GF2to16_table, *new_iter);

                  coset_trace(GF2to16_table);

                  calc_roots(GF2to16_table,    GF2to16_table.field_size,
1, GF2to16_table.extn_degree);
                  new_iter++;

            }//end inner while

      my_iter++;
      }//end while
*/
//**********end iterative run


}//end main

//circle_add takes two elements that you want to circle_add and the
field that they are in and returns the results.
int_type  circle_add(  int_type  left,  int_type  right,  galois_table
curr_field)
{
      int_type    first_temp_left,    second_temp_left,    result_left,
first_temp_rt, second_temp_rt, result_rt;

      if (left==STAR)
            return right;
      else if (right==STAR)
            return left;
      else if (left==right)
            return STAR;
      else //neither have STAR as content and they arent equal
      {
```

```
            //fill temp regs
            first_temp_left = curr_field.prev->curr[left][0];
            first_temp_rt = curr_field.prev->curr[left][1];
            second_temp_left = curr_field.prev->curr[right][0];
            second_temp_rt = curr_field.prev->curr[right][1];

            //circle add component wise, starting with left
            result_left = circle_add(first_temp_left, second_temp_left,
*(curr_field.prev));

            //circle add component wise, now with right
            result_rt   =   circle_add(first_temp_rt,   second_temp_rt,
*(curr_field.prev));

            //need to find row in the prev field where result_left and
result_rt are located
            for(int j = 0; j < (curr_field.prev_field_size - 1); j++)
            {
                  if   ((curr_field.prev->curr[j][0]==result_left)   &&
(curr_field.prev->curr[j][1]==result_rt))
                        return j;
            }//end for
      }//end else

}//end circle add

//build_extn_field   takes   galois_table   struct   and   the   constant
coefficient of the primitive
// polynomial that you want to build the field with and runs through
the Galois shift
//register in order to build the field. It puts each nonzero element in
the table.
void build_extn_field(galois_table &table, int_type multiplier)
{


      dump_regs(table, 0);

      for (int row = 1; row < table.field_size - 1; row++)
      {
            //printf("build_etn_field::%d\n", row);
            int first_temp_left, second_temp_left,
                  result_left, first_temp_rt, second_temp_rt,
                  result_rt;

            curr_left_reg   =   circle_add(old_left_reg,   old_rt_reg,
table);

            //now calculate curr_rt_reg
            if (old_left_reg == STAR)
                  curr_rt_reg = STAR;
            else  if((old_left_reg  >=  pow(2,  sanity_check_num))  &&
(multiplier >=pow(2, sanity_check_num)))
            {
                  fprintf(stderr, "Numbers are out of range. Need more
bits in the machine.\n");
                  exit(0);
```

```
            }

            else
                  curr_rt_reg   =   (old_left_reg   +   multiplier)   %
(table.prev_field_size - 1);

            //put contents of regs in table
            dump_regs(table, row);

            //update reg values
            old_left_reg = curr_left_reg;
            old_rt_reg = curr_rt_reg;

            //check for root position
            if((curr_left_reg == STAR) && (curr_rt_reg == 1))
            {
                  //printf("found root at: %d\n", row);
                  table.root_position = row;
            //    printf("root        position        is:        %d\n",
table.root_position);
            }


      }//end for

}

//coset_trace goes through each constant coefficient to determine if
x^2+x+constant is
//irreducible or not by determining the trace of the constant. If the
polynomial is irreducible,
// it determines the order of the constant in the previous Galois field
to see if it has full
//order and therefore the polynomial is primitive. If the polynomial is
primitive, then
//coset_trace determines the other elements of the coset and puts them
all in a vector.
//coset_trace returns this vector in case you want to iteratively run
through the fields.
NumVector coset_trace(galois_table table)
{
      int_type ** trace_table;
      int_type * coset_array;
      trace_table =     new   int_type*[table.extn_degree];//trace_table
holds a representation of each element in a coset that a particular
constant coefficient also belongs to
      coset_array = new int_type[table.field_size - 1]; //coset_array
holds all nonzero elements of a field and is used to keep track of
whether or not x^2+x+element is irreducible and primitive
      int_type  left_trace = 0;//left trace is the left part of the
representation of each element in a coset, circle_added together
      int_type  rt_trace = 0;//right trace is the right part of the
representation of each element in a coset, circle_added together
      int_type order;//the order of a particular constant coefficient
in the previous galois field
```

```
      NumVector ret;//the vector that coset_trace returns; holds the
constant coefficients for each x^2+x+constant that is primitive over
the current galois field

      //make sure there's enough memory for the tables
      if (!trace_table || !coset_array)
      {
            printf("coset_trace::Error allocating memory for trace_tale
or coset_array\n");
            exit(0);
      }

      for (int i = 0; i < table.extn_degree; i++) //NO subtracting from
extn_degree
      {
            trace_table[i] = new int_type[2];
            if (trace_table[i] == NULL)
            {
                  printf("Error  allocating  trace-table  .  i  =  %d
exiting\n", i);
                  exit(0);
            }
      }

      //initialize coset array
      for (int k = 0; k < table.field_size - 1; k++)
            coset_array[k] = 0;

      //find next coset_rep, fill trace_table, print coset,
      //determine trace, print trace..

      for (int j = 0; j < table.field_size - 1; j++)
      {
            //printf("J:[%d/%d]\n ", j, table.field_size - 1);
            if  (coset_array[j]==0)//coset_array[j]==0  means  it  hasn't
been checked yet
            {
//                printf("coset rep is: C%d\n", j);
                  coset_array[j]  =1;//coset_array[j]==1  means  it  has
been checked for being irred/red
                  trace_table[0][0] = table.curr[j][0];
                  trace_table[0][1] = table.curr[j][1];

                  //fill trace table
                  for (int k = 1; k <table.extn_degree; k++)
                  {
                        //sanity check for bits
                        if(high_bit_pos(j)  +  high_bit_pos((2<<(k-1)))
>= sanity_check_num + 1)
                        {
                              fprintf(stderr,  "numbers  of  out  range.
Need more bits in the machine");
                              exit(0);
                        }

                        //offset is the row in the current galois field
table where a coset member's representation is located
```

```
                        int offset =   (j*     ( 2 << (k-1)) ) % (
table.field_size-1)  ;
                        //printf("\tK:                    [%d/%d]\n",k,
table.extn_degree);
//                      printf("C%d is in this coset \n", offset);
                        coset_array[ offset   ] = 2;//coset_array[j]==2
means that it is a coset member in a coset where one of the members
whose trace has already been computed. Since elements in the coset are
either all red/irred, we only need to check one.
                        trace_table[k][0] = table.curr[offset][0];
                        trace_table[k][1] = table.curr[offset][1];
                        //printf("\tK:                  --[%d/%d]\n",k,
table.extn_degree);
                  }//end for (k=1..extn_degre)..

                  //determine trace
                  left_trace                                          =
circle_add(trace_table[0][0],trace_table[1][0], table);
                  for (int n =2; n < table.extn_degree; n++)
                  {
                        left_trace       =       circle_add(left_trace,
trace_table[n][0], table);
                  }

                  rt_trace                                            =
circle_add(trace_table[0][1],trace_table[1][1], table);
                  for (int n =2; n < table.extn_degree; n++)
                  {
                        rt_trace         =         circle_add(rt_trace,
trace_table[n][1], table);
                  }

                  //print trace
                  //printf("trace of c %d is: %d %d\n", j, left_trace,
rt_trace);

                  if((left_trace==STAR) && (rt_trace==STAR))
//                      printf("x^2 + x + c^%d is reducible\n\n", j);
                        ;//ghetto  hack,  but  oh  well.  semicolon  NOT
needed if the print statement is not commented out.
                  else if ((left_trace ==STAR) && (rt_trace ==0))
                  {
//                      printf("x^2 + x + c^%d is irreducible\n", j);
                        order = check_order(j, table);
                        if (order == (table.prev_field_size - 1))
                        {
                              //put j in the vector
                              ret.push_back(j);

                              for  (int  k  =  1;  k  <table.extn_degree;
k++)
                              {
                                    //sanity check the bits
                                    if        (high_bit_pos(j)        +
high_bit_pos((2<<(k-1))) >= sanity_check_num + 1)
                                    {
```

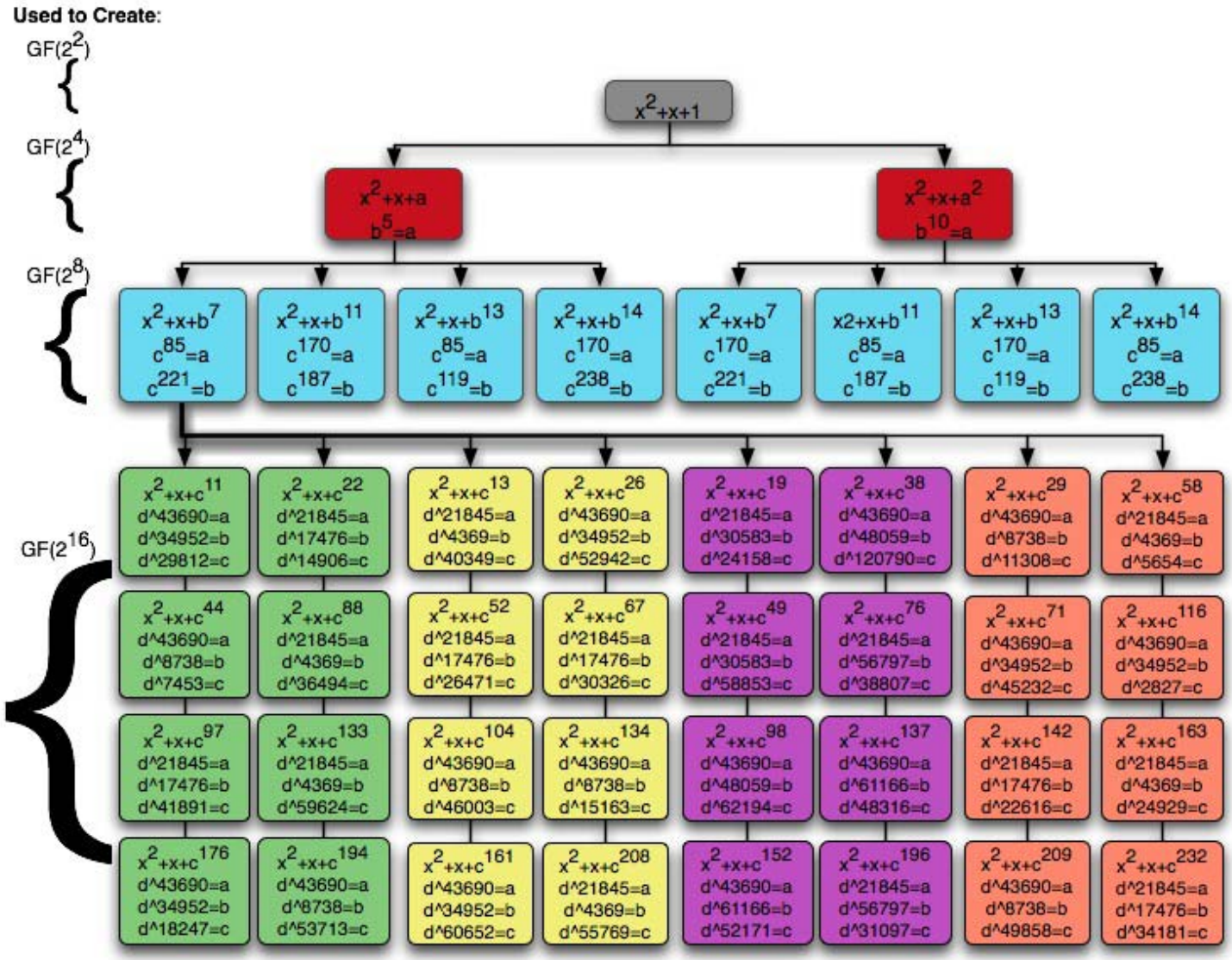
```
                                          fprintf(stderr,  "Numbers  out
of range. Need more bits in the machine.");
                                          exit(0);
                                    }
                                    int offset =  (j*   ( 2 << (k-1)) )
% ( table.field_size-1)  ;

                                    //put coset members in here
                                    ret.push_back(offset);
                                    //printf("C%d is in this coset \n",
offset);
                              }//end for

//                            printf("x^2    +    x    +    c^%d    is
primitive.\n\n", j);

                        }//end if
                        else
//                            printf("x^2 + x + c^%d is not primitive
with period %d\n\n", j, order);
                        ;//ghetto hack again. semicolon NOT needed if
the print statement is not commented out.
                  }//end else if

                  else//Trace  should  only  be  one  of  two  things.  if
trace is wrong, this will catch it and give you an error message.
                  {
                        printf("error. trace is *not* correct\n");
                        printf("left  trace  =  %d,  rt_trace  =  %d\n",
left_trace, rt_trace);
                  }

            }//end if
      }//end for
      printf("\n");
      return ret;

}//end coset_trace

//print_table  prints  the  elements  of  the  field  whose  galois_table
struct it is passed
void print_table (galois_table table)
{
      int i;
      //fprintf(stderr,"      |C    |\n\n");
      for (int i = 0; i < table.field_size - 1; i++)
      {
            printf("root^%d: ", i);
            if (table.curr[i][0] == STAR)
                  printf("STAR,");
            else
                  printf("%4d,", table.curr[i][0]);

            if (table.curr[i][1] == STAR)
                  printf("STAR\n");
            else
                  printf("%4d\n", table.curr[i][1]);
```

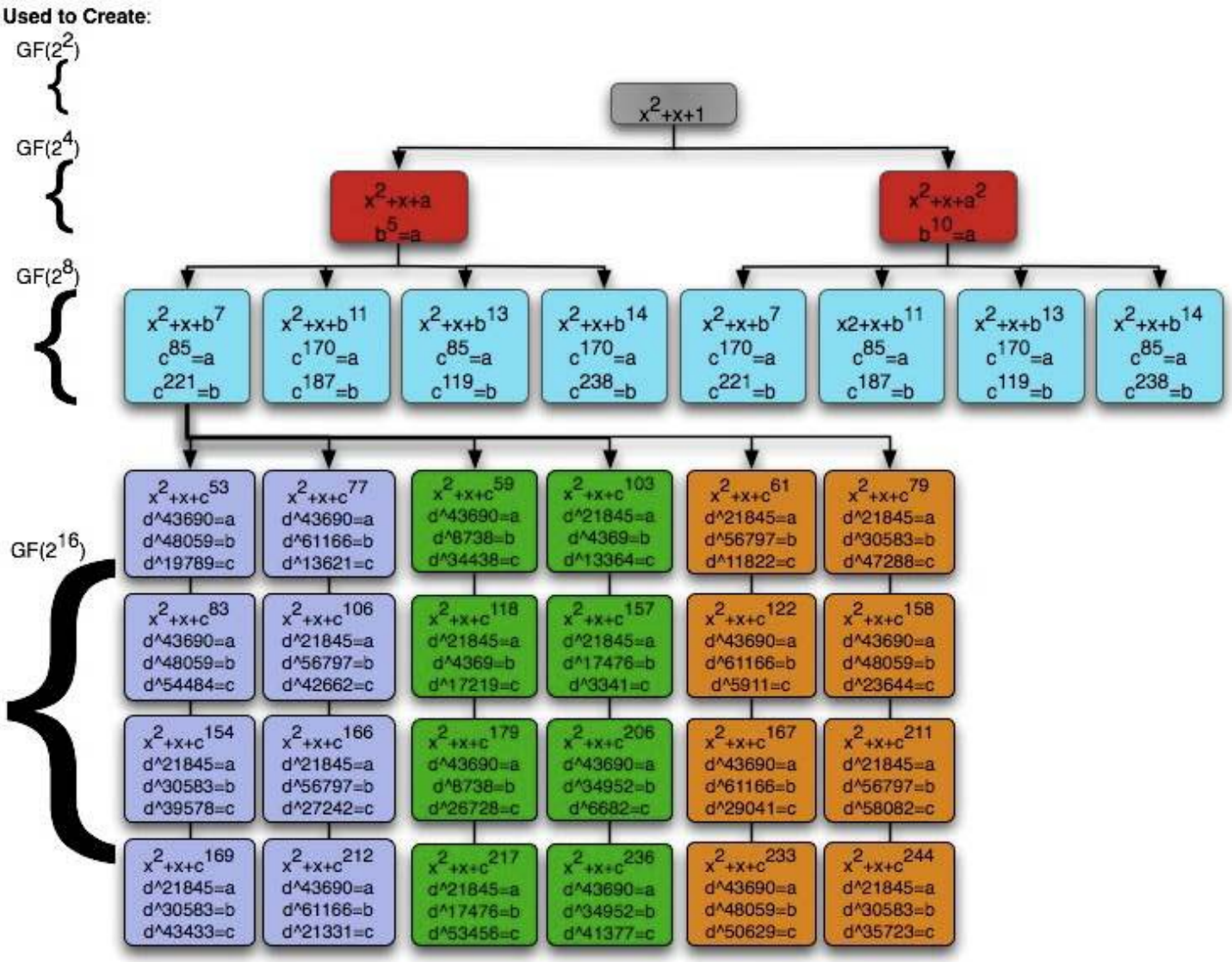
```
	}

}//end print_table

//check_order takes a field element and finds its order in the previous
galois field. returns the order. check_order assumes the element passed
to it is not 0!!!
int_type check_order(int_type number, galois_table table)
{
	//find order of number in prev_field
	// *** assumes number is not 0! ***
	int_type test = number;
	int_type order = 1;

	while(test != 0)
	{
		test = (test + number) % (table.prev_field_size - 1);
		order++;
	}//end while

	return order;


}//end check_order

//calc_roots prints out the location in the current field of all
previously seen roots
void calc_roots(galois_table table, int_type top_field_size, int_type
prev_root_pos, int_type times)
{
	int_type curr_root_pos;

	if(high_bit_pos(prev_root_pos)                                    +
high_bit_pos(table.root_position) >= sanity_check_num +1 )
	{
		fprintf(stderr, "Numbers of out range. Need more bits on
the machine.");
		exit(0);
	}

	curr_root_pos   =   (prev_root_pos   *   table.root_position)   %
(top_field_size - 1);

//	printf("prev_root_pos   is:   %d   table.root_position   is:   %d
top_field_size   is:   %d   curr_root_pos   is:   %d\n",   prev_root_pos,
table.root_position, top_field_size, curr_root_pos);

	if(times == 4)//last time
	{
		printf("position  of  root  from  degree  %d  extension  is:
%d\n", times, curr_root_pos);
	}

	else if (times > 4)//not last time. so waant to print stuff and
call function again
	{
```

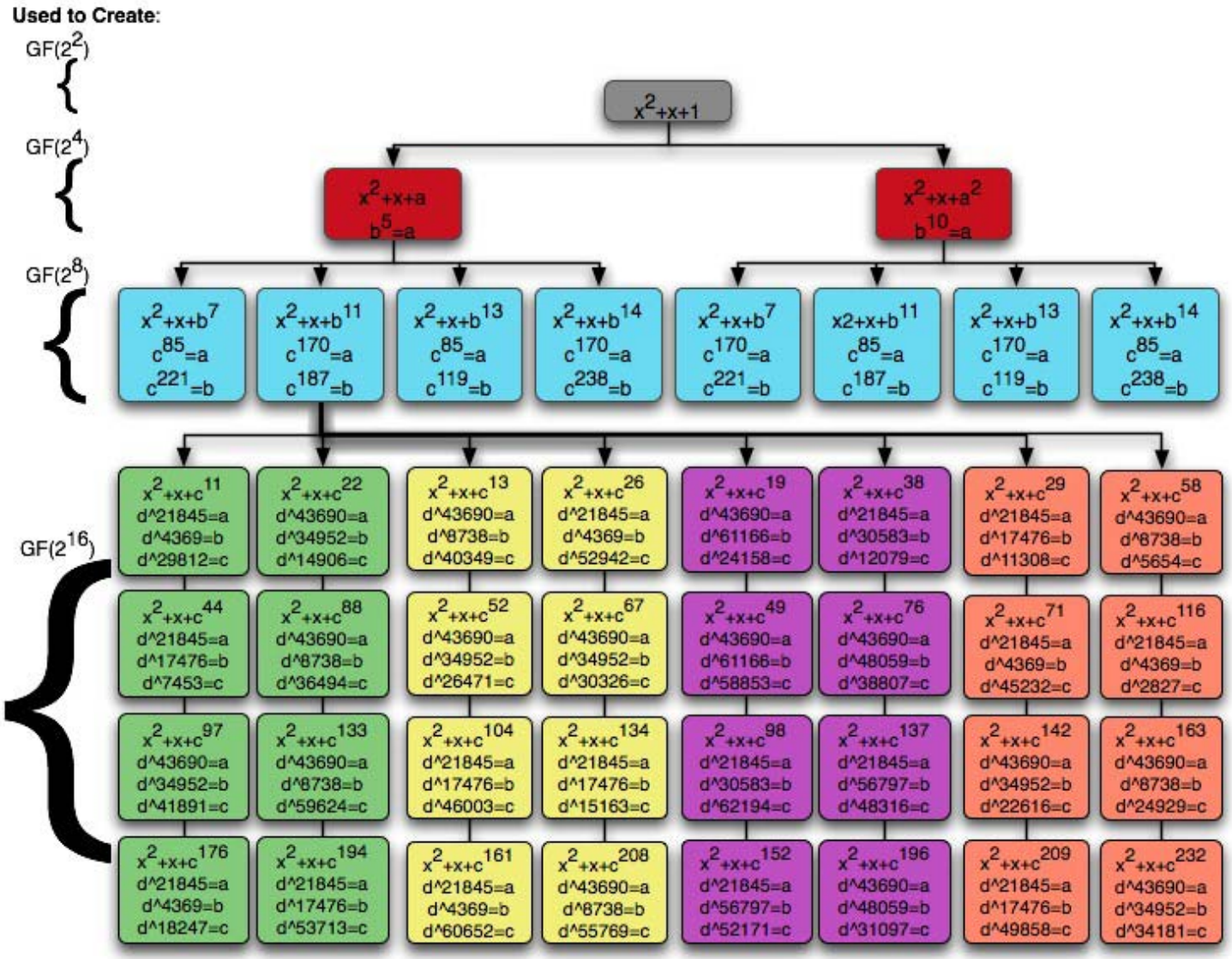
```
            printf("position of root from degree %d extension is:
%d\n", times, curr_root_pos);
            calc_roots(*(table.prev), top_field_size, curr_root_pos,
(times/2));
      }



}//end calc_roots

//high_bit_pos takes a number and returns the highest bit position
needed to represent the number in binary
int_type high_bit_pos(int_type number)
{
      int_type count = 0;
      for(int_type temp = number; temp>1; temp = temp >>1)
      {
            count++;
      }
      return count;
}//end high_bit_num
```

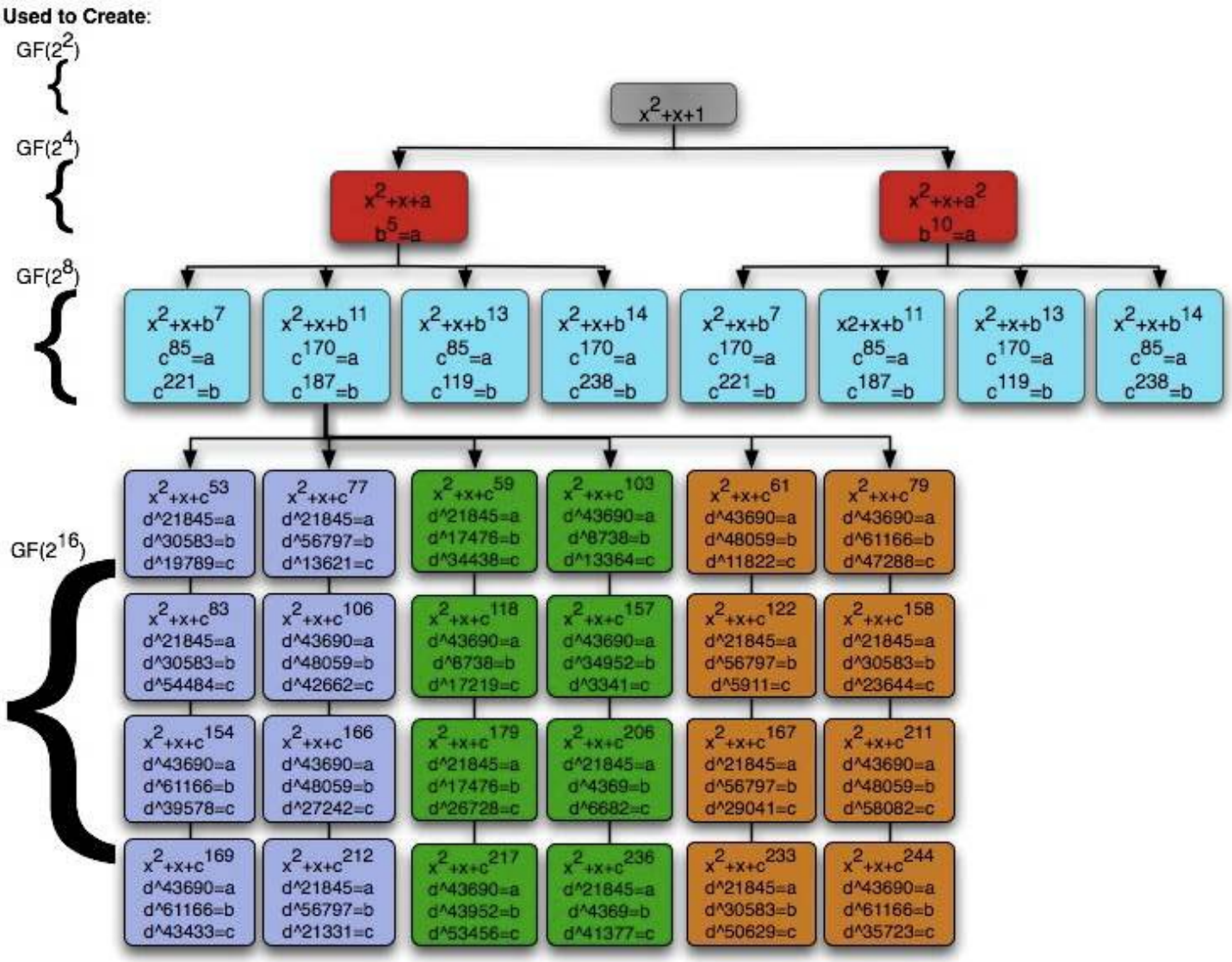
APPENDIX E. CHARTS

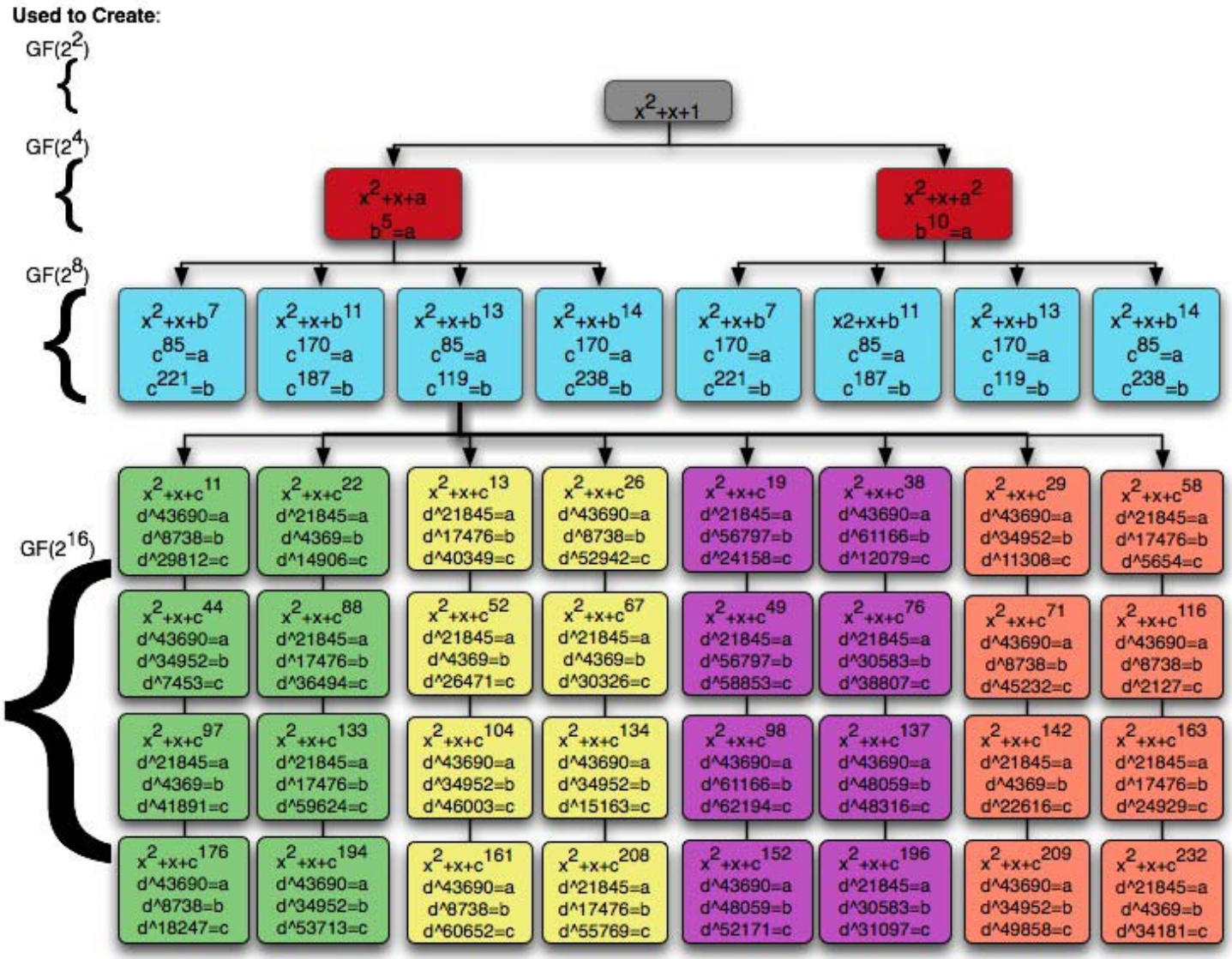
Appendix E contains the charts that are a visual representation of the fields generated by polynomials of the form $x^2 + x + \alpha^i$ for a primitive element α. These charts also show which powers of the current primitive root are equal to previous primitive roots.

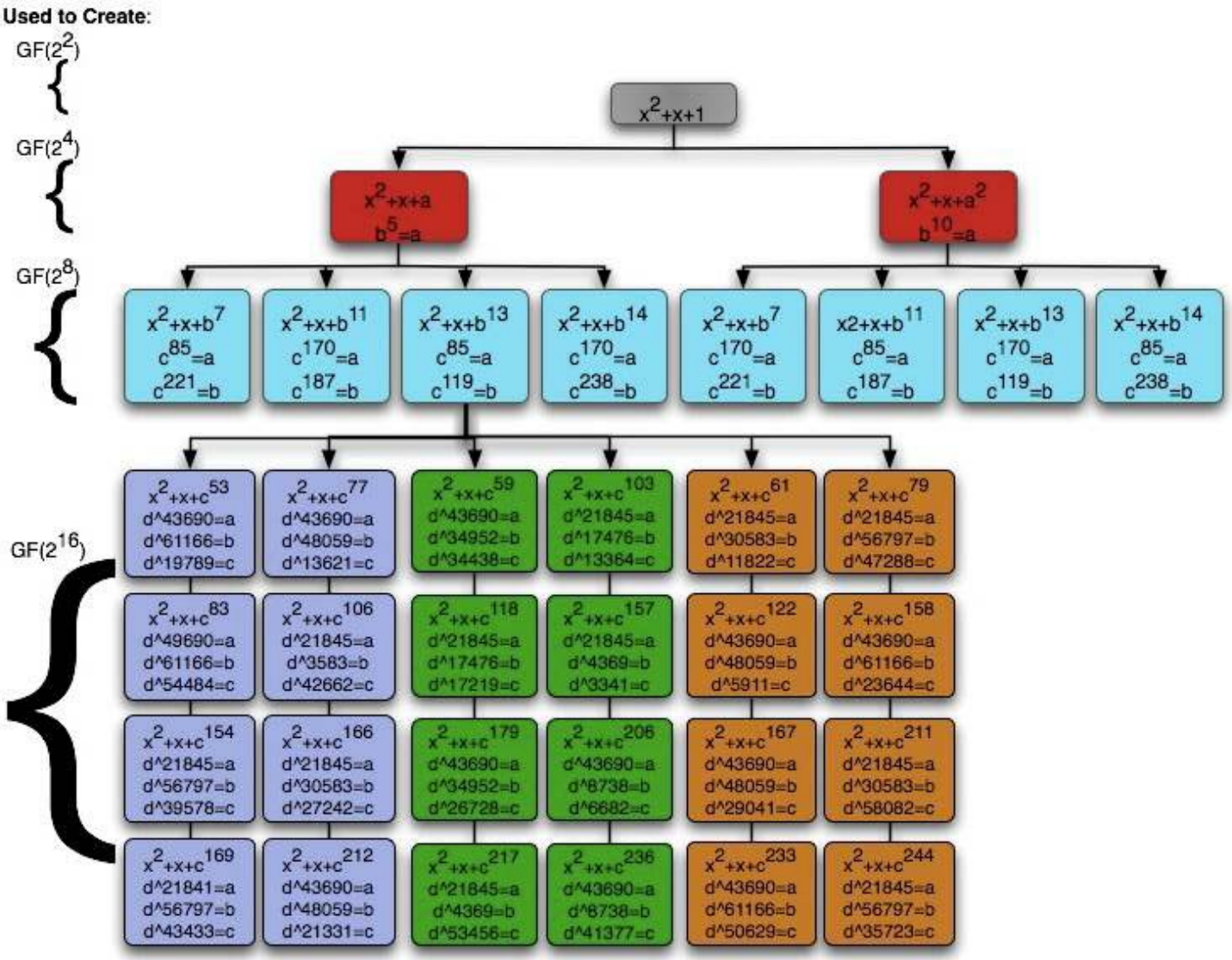

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^11
d^43690=a
d^34952=b
d^29812=c
x^2+x+c^22
d^21845=a
d^17476=b
d^14906=c
x^2+x+c^13
d^21845=a
d^4369=b
d^40349=c
x^2+x+c^26
d^43690=a
d^34952=b
d^52942=c
x^2+x+c^19
d^21845=a
d^30583=b
d^24158=c
x^2+x+c^38
d^43690=a
d^48059=b
d^120790=c
x^2+x+c^29
d^43690=a
d^8738=b
d^11308=c
x^2+x+c^58
d^21845=a
d^4369=b
d^5654=c
x^2+x+c^44
d^43690=a
d^8738=b
d^7453=c
x^2+x+c^88
d^21845=a
d^4369=b
d^36494=c
x^2+x+c^52
d^21845=a
d^17476=b
d^26471=c
x^2+x+c^67
d^21845=a
d^17476=b
d^30326=c
x^2+x+c^49
d^21845=a
d^30583=b
d^58853=c
x^2+x+c^76
d^21845=a
d^56797=b
d^38807=c
x^2+x+c^71
d^43690=a
d^34952=b
d^45232=c
x^2+x+c^116
d^43690=a
d^34952=b
d^2827=c
x^2+x+c^97
d^21845=a
d^17476=b
d^41891=c
x^2+x+c^133
d^21845=a
d^4369=b
d^59624=c
x^2+x+c^104
d^43690=a
d^8738=b
d^46003=c
x^2+x+c^134
d^43690=a
d^8738=b
d^15163=c
x^2+x+c^98
d^43690=a
d^48059=b
d^62194=c
x^2+x+c^137
d^43690=a
d^61166=b
d^48316=c
x^2+x+c^142
d^21845=a
d^17476=b
d^22616=c
x^2+x+c^163
d^21845=a
d^4369=b
d^24929=c
x^2+x+c^176
d^43690=a
d^34952=b
d^18247=c
x^2+x+c^194
d^43690=a
d^8738=b
d^53713=c
x^2+x+c^161
d^43690=a
d^34952=b
d^60652=c
x^2+x+c^208
d^21845=a
d^4369=b
d^55769=c
x^2+x+c^152
d^43690=a
d^61166=b
d^52171=c
x^2+x+c^196
d^21845=a
d^56797=b
d^31097=c
x^2+x+c^209
d^43690=a
d^8738=b
d^49858=c
x^2+x+c^232
d^21845=a
d^17476=b
d^34181=c

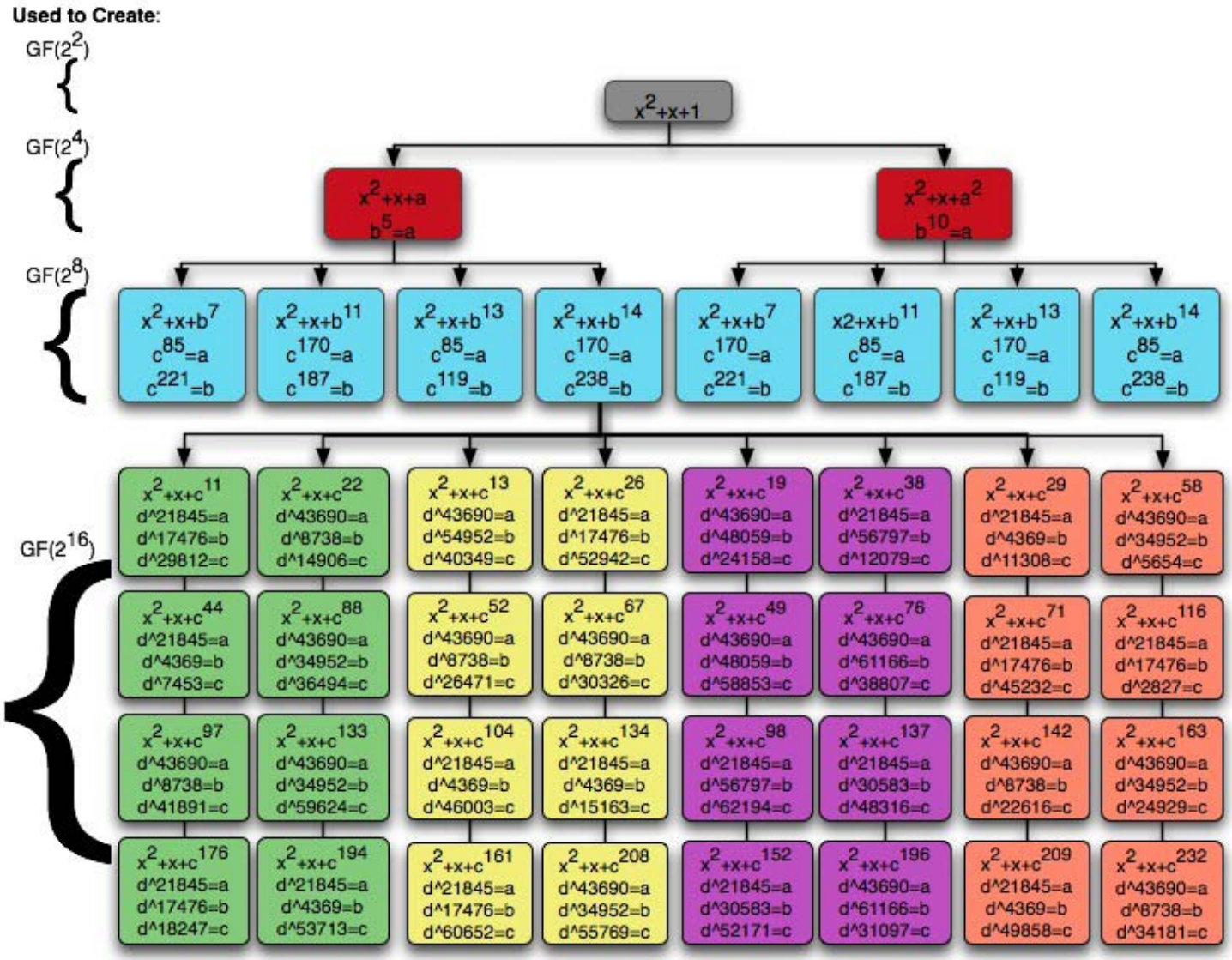

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^53
d^43690=a
d^48059=b
d^19789=c
x^2+x+c^77
d^43690=a
d^61166=b
d^13621=c
x^2+x+c^59
d^43690=a
d^8738=b
d^34438=c
x^2+x+c^103
d^21845=a
d^4369=b
d^13364=c
x^2+x+c^61
d^21845=a
d^56797=b
d^11822=c
x^2+x+c^79
d^21845=a
d^30583=b
d^47288=c
x^2+x+c^83
d^43690=a
d^48059=b
d^54484=c
x^2+x+c^106
d^21845=a
d^56797=b
d^42662=c
x^2+x+c^118
d^21845=a
d^4369=b
d^17219=c
x^2+x+c^157
d^21845=a
d^17476=b
d^3341=c
x^2+x+c^122
d^43690=a
d^61166=b
d^5911=c
x^2+x+c^158
d^43690=a
d^48059=b
d^23644=c
x^2+x+c^154
d^21845=a
d^30583=b
d^39578=c
x^2+x+c^166
d^21845=a
d^56797=b
d^27242=c
x^2+x+c^179
d^43690=a
d^8738=b
d^26728=c
x^2+x+c^206
d^43690=a
d^34952=b
d^6682=c
x^2+x+c^167
d^43690=a
d^61166=b
d^29041=c
x^2+x+c^211
d^21845=a
d^56797=b
d^58082=c
x^2+x+c^169
d^21845=a
d^30583=b
d^43433=c
x^2+x+c^212
d^43690=a
d^61166=b
d^21331=c
x^2+x+c^217
d^21845=a
d^17476=b
d^53456=c
x^2+x+c^236
d^43690=a
d^34952=b
d^41377=c
x^2+x+c^233
d^43690=a
d^48059=b
d^50629=c
x^2+x+c^244
d^21845=a
d^30583=b
d^35723=c

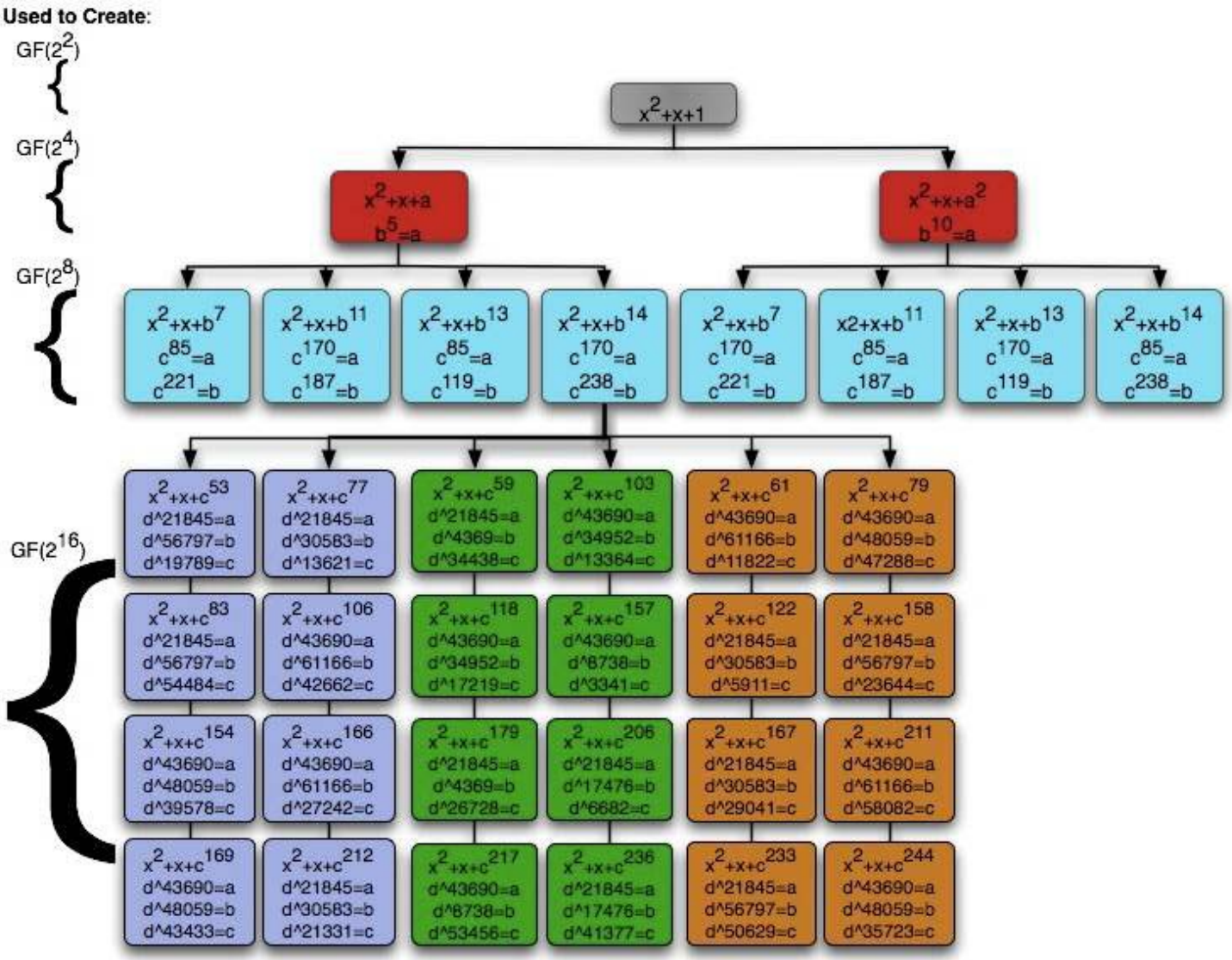

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^11
d^21845=a
d^4369=b
d^29812=c
x^2+x+c^22
d^43690=a
d^34952=b
d^14906=c
x^2+x+c^13
d^43690=a
d^8738=b
d^40349=c
x^2+x+c^26
d^21845=a
d^4369=b
d^52942=c
x^2+x+c^19
d^43690=a
d^61166=b
d^24158=c
x^2+x+c^38
d^21845=a
d^30583=b
d^12079=c
x^2+x+c^29
d^21845=a
d^17476=b
d^11308=c
x^2+x+c^58
d^43690=a
d^8738=b
d^5654=c
x^2+x+c^44
d^21845=a
d^17476=b
d^7453=c
x^2+x+c^88
d^43690=a
d^8738=b
d^36494=c
x^2+x+c^52
d^43690=a
d^34952=b
d^26471=c
x^2+x+c^67
d^43690=a
d^34952=b
d^30326=c
x^2+x+c^49
d^43690=a
d^61166=b
d^58853=c
x^2+x+c^76
d^43690=a
d^48059=b
d^38807=c
x^2+x+c^71
d^21845=a
d^4369=b
d^45232=c
x^2+x+c^116
d^21845=a
d^4369=b
d^2827=c
x^2+x+c^97
d^43690=a
d^34952=b
d^41891=c
x^2+x+c^133
d^43690=a
d^8738=b
d^59624=c
x^2+x+c^104
d^21845=a
d^17476=b
d^46003=c
x^2+x+c^134
d^21845=a
d^17476=b
d^15163=c
x^2+x+c^98
d^21845=a
d^30583=b
d^62194=c
x^2+x+c^137
d^21845=a
d^56797=b
d^48316=c
x^2+x+c^142
d^43690=a
d^34952=b
d^22616=c
x^2+x+c^163
d^43690=a
d^8738=b
d^24929=c
x^2+x+c^176
d^21845=a
d^4369=b
d^18247=c
x^2+x+c^194
d^21845=a
d^17476=b
d^53713=c
x^2+x+c^161
d^21845=a
d^4369=b
d^60652=c
x^2+x+c^208
d^43690=a
d^8738=b
d^55769=c
x^2+x+c^152
d^21845=a
d^56797=b
d^52171=c
x^2+x+c^196
d^43690=a
d^48059=b
d^31097=c
x^2+x+c^209
d^21845=a
d^17476=b
d^49858=c
x^2+x+c^232
d^43690=a
d^34952=b
d^34181=c

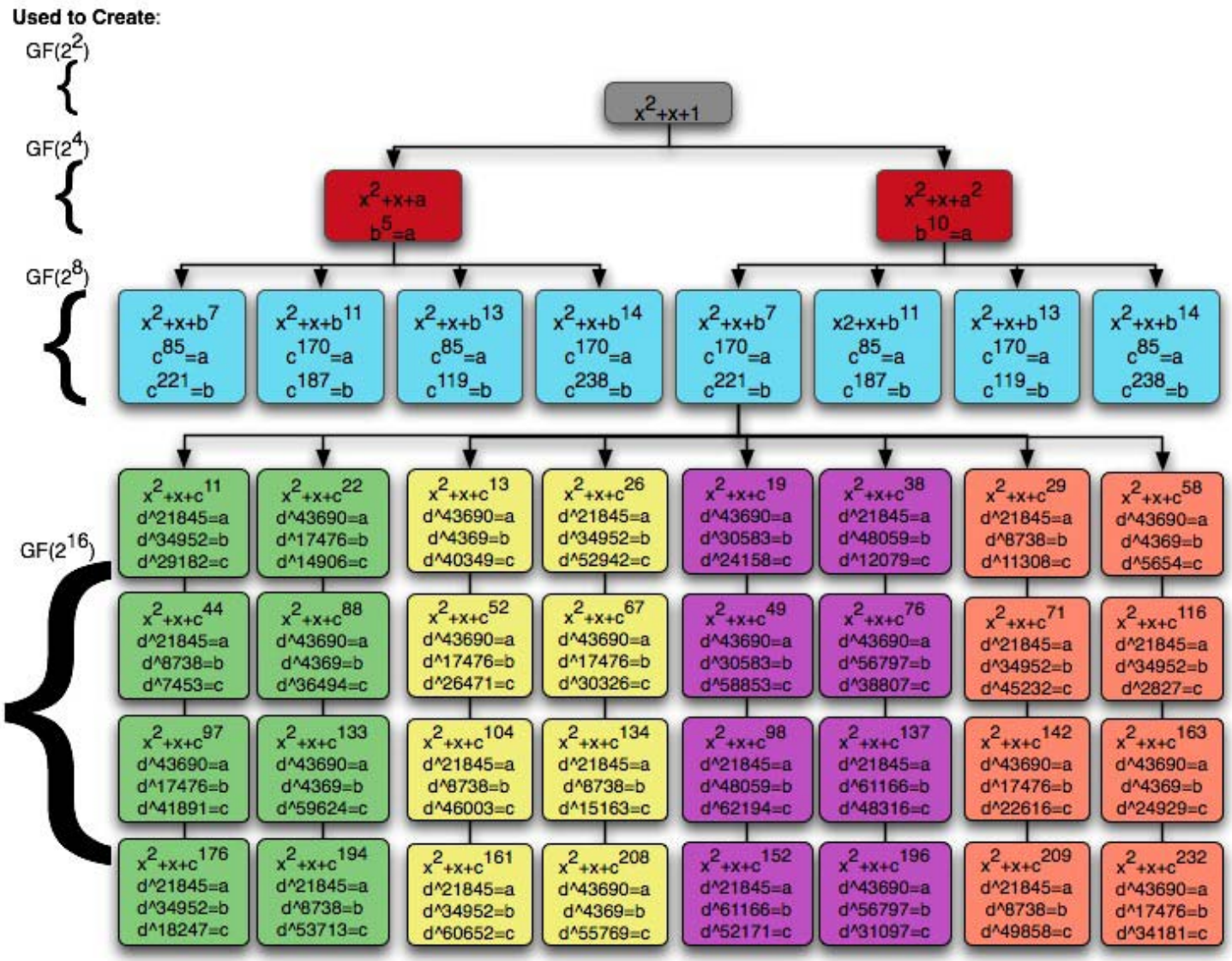

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^53
d^21845=a
d^30583=b
d^19789=c
x^2+x+c^77
d^21845=a
d^56797=b
d^13621=c
x^2+x+c^59
d^21845=a
d^17476=b
d^34438=c
x^2+x+c^103
d^43690=a
d^8738=b
d^13364=c
x^2+x+c^61
d^43690=a
d^48059=b
d^11822=c
x^2+x+c^79
d^43690=a
d^61166=b
d^47288=c
x^2+x+c^83
d^21845=a
d^30583=b
d^54484=c
x^2+x+c^106
d^43690=a
d^48059=b
d^42662=c
x^2+x+c^118
d^43690=a
d^6738=b
d^17219=c
x^2+x+c^157
d^43690=a
d^34952=b
d^3341=c
x^2+x+c^122
d^21845=a
d^56797=b
d^5911=c
x^2+x+c^158
d^21845=a
d^30583=b
d^23644=c
x^2+x+c^154
d^43690=a
d^61166=b
d^39578=c
x^2+x+c^166
d^43690=a
d^48059=b
d^27242=c
x^2+x+c^179
d^21845=a
d^17476=b
d^26728=c
x^2+x+c^206
d^21845=a
d^4369=b
d^6682=c
x^2+x+c^167
d^21845=a
d^56797=b
d^29041=c
x^2+x+c^211
d^43690=a
d^48059=b
d^58082=c
x^2+x+c^169
d^43690=a
d^61166=b
d^43433=c
x^2+x+c^212
d^21845=a
d^56797=b
d^21331=c
x^2+x+c^217
d^43690=a
d^43952=b
d^53456=c
x^2+x+c^236
d^21845=a
d^4369=b
d^41377=c
x^2+x+c^233
d^21845=a
d^30583=b
d^50629=c
x^2+x+c^244
d^43690=a
d^61166=b
d^35723=c

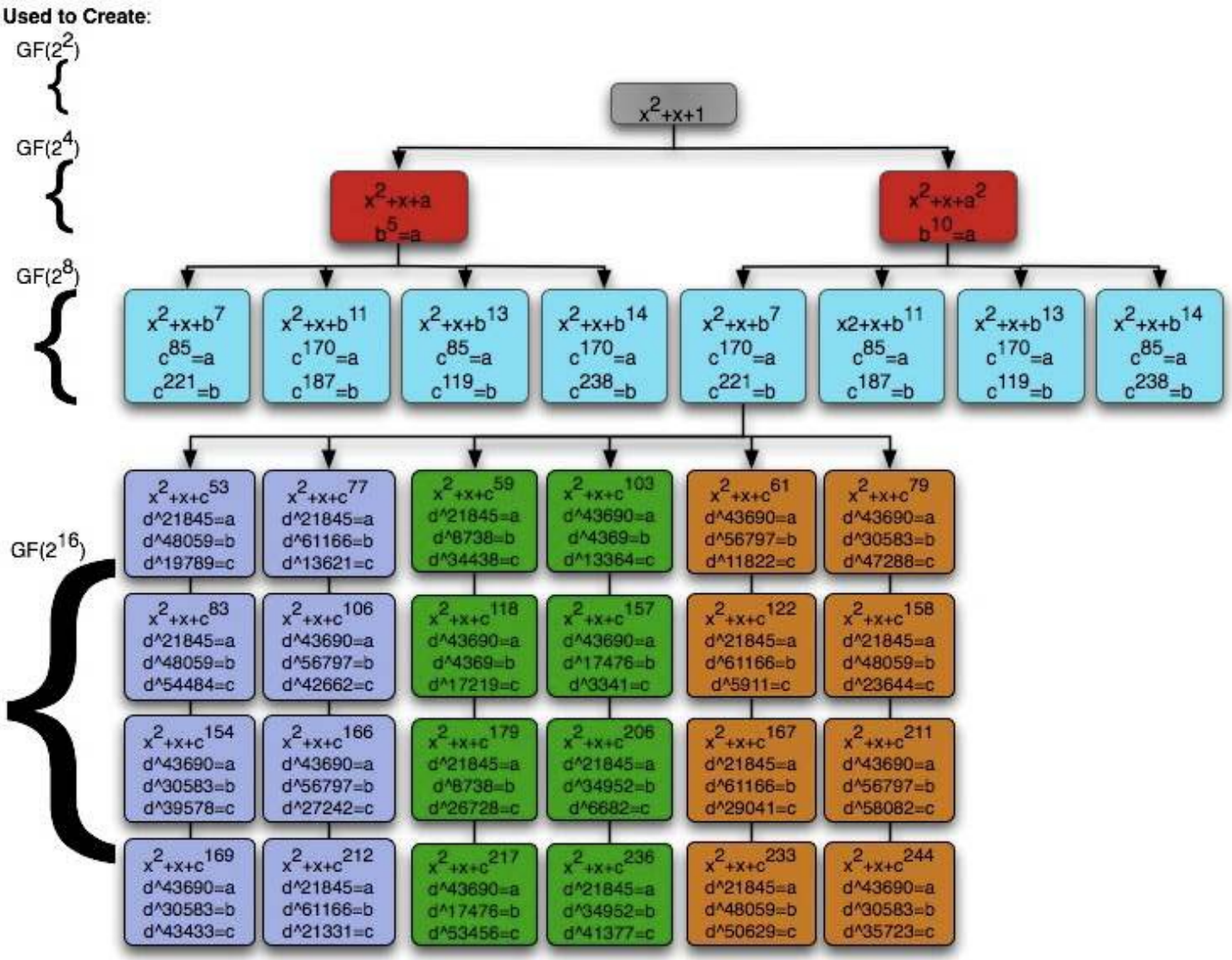

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^11
d^43690=a
d^8738=b
d^29812=c
x^2+x+c^22
d^21845=a
d^4369=b
d^14906=c
x^2+x+c^13
d^21845=a
d^17476=b
d^40349=c
x^2+x+c^26
d^43690=a
d^8738=b
d^52942=c
x^2+x+c^19
d^21845=a
d^56797=b
d^24158=c
x^2+x+c^38
d^43690=a
d^61166=b
d^12079=c
x^2+x+c^29
d^43690=a
d^34952=b
d^11308=c
x^2+x+c^58
d^21845=a
d^17476=b
d^5654=c
x^2+x+c^44
d^43690=a
d^34952=b
d^7453=c
x^2+x+c^88
d^21845=a
d^17476=b
d^36494=c
x^2+x+c^52
d^21845=a
d^4369=b
d^26471=c
x^2+x+c^67
d^21845=a
d^4369=b
d^30326=c
x^2+x+c^49
d^21845=a
d^56797=b
d^58853=c
x^2+x+c^76
d^21845=a
d^30583=b
d^38807=c
x^2+x+c^71
d^43690=a
d^8738=b
d^45232=c
x^2+x+c^116
d^43690=a
d^8738=b
d^2127=c
x^2+x+c^97
d^21845=a
d^4369=b
d^41891=c
x^2+x+c^133
d^21845=a
d^17476=b
d^59624=c
x^2+x+c^104
d^43690=a
d^34952=b
d^46003=c
x^2+x+c^134
d^43690=a
d^34952=b
d^15163=c
x^2+x+c^98
d^43690=a
d^61166=b
d^62194=c
x^2+x+c^137
d^43690=a
d^48059=b
d^48316=c
x^2+x+c^142
d^21845=a
d^4369=b
d^22616=c
x^2+x+c^163
d^21845=a
d^17476=b
d^24929=c
x^2+x+c^176
d^43690=a
d^8738=b
d^18247=c
x^2+x+c^194
d^43690=a
d^34952=b
d^53713=c
x^2+x+c^161
d^43690=a
d^8738=b
d^60652=c
x^2+x+c^208
d^21845=a
d^17476=b
d^55769=c
x^2+x+c^152
d^43690=a
d^48059=b
d^52171=c
x^2+x+c^196
d^21845=a
d^30583=b
d^31097=c
x^2+x+c^209
d^43690=a
d^34952=b
d^49858=c
x^2+x+c^232
d^21845=a
d^4369=b
d^34181=c

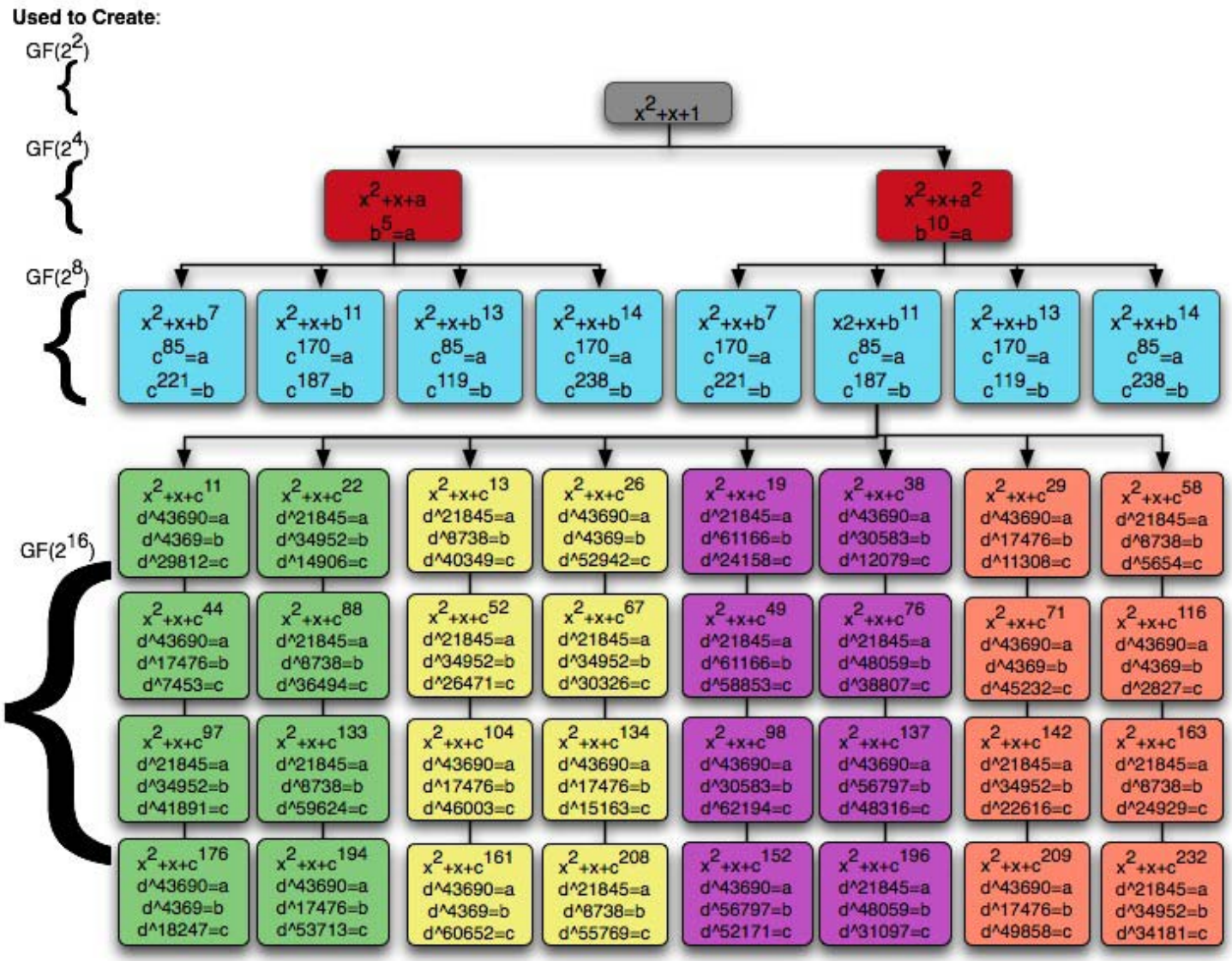

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^53
d^43690=a
d^61166=b
d^19789=c
x^2+x+c^77
d^43690=a
d^48059=b
d^13621=c
x^2+x+c^59
d^43690=a
d^34952=b
d^34438=c
x^2+x+c^103
d^21845=a
d^17476=b
d^13364=c
x^2+x+c^61
d^21845=a
d^30583=b
d^11822=c
x^2+x+c^79
d^21845=a
d^56797=b
d^47288=c
x^2+x+c^83
d^49690=a
d^61166=b
d^54484=c
x^2+x+c^106
d^21845=a
d^3583=b
d^42662=c
x^2+x+c^118
d^21845=a
d^17476=b
d^17219=c
x^2+x+c^157
d^21845=a
d^4369=b
d^3341=c
x^2+x+c^122
d^43690=a
d^48059=b
d^5911=c
x^2+x+c^158
d^43690=a
d^61166=b
d^23644=c
x^2+x+c^154
d^21845=a
d^56797=b
d^39578=c
x^2+x+c^166
d^21845=a
d^30583=b
d^27242=c
x^2+x+c^179
d^43690=a
d^34952=b
d^26728=c
x^2+x+c^206
d^43690=a
d^8738=b
d^6682=c
x^2+x+c^167
d^43690=a
d^48059=b
d^29041=c
x^2+x+c^211
d^21845=a
d^30583=b
d^58082=c
x^2+x+c^169
d^21841=a
d^56797=b
d^43433=c
x^2+x+c^212
d^43690=a
d^48059=b
d^21331=c
x^2+x+c^217
d^21845=a
d^4369=b
d^53456=c
x^2+x+c^236
d^43690=a
d^8738=b
d^41377=c
x^2+x+c^233
d^43690=a
d^61166=b
d^50629=c
x^2+x+c^244
d^21845=a
d^56797=b
d^35723=c

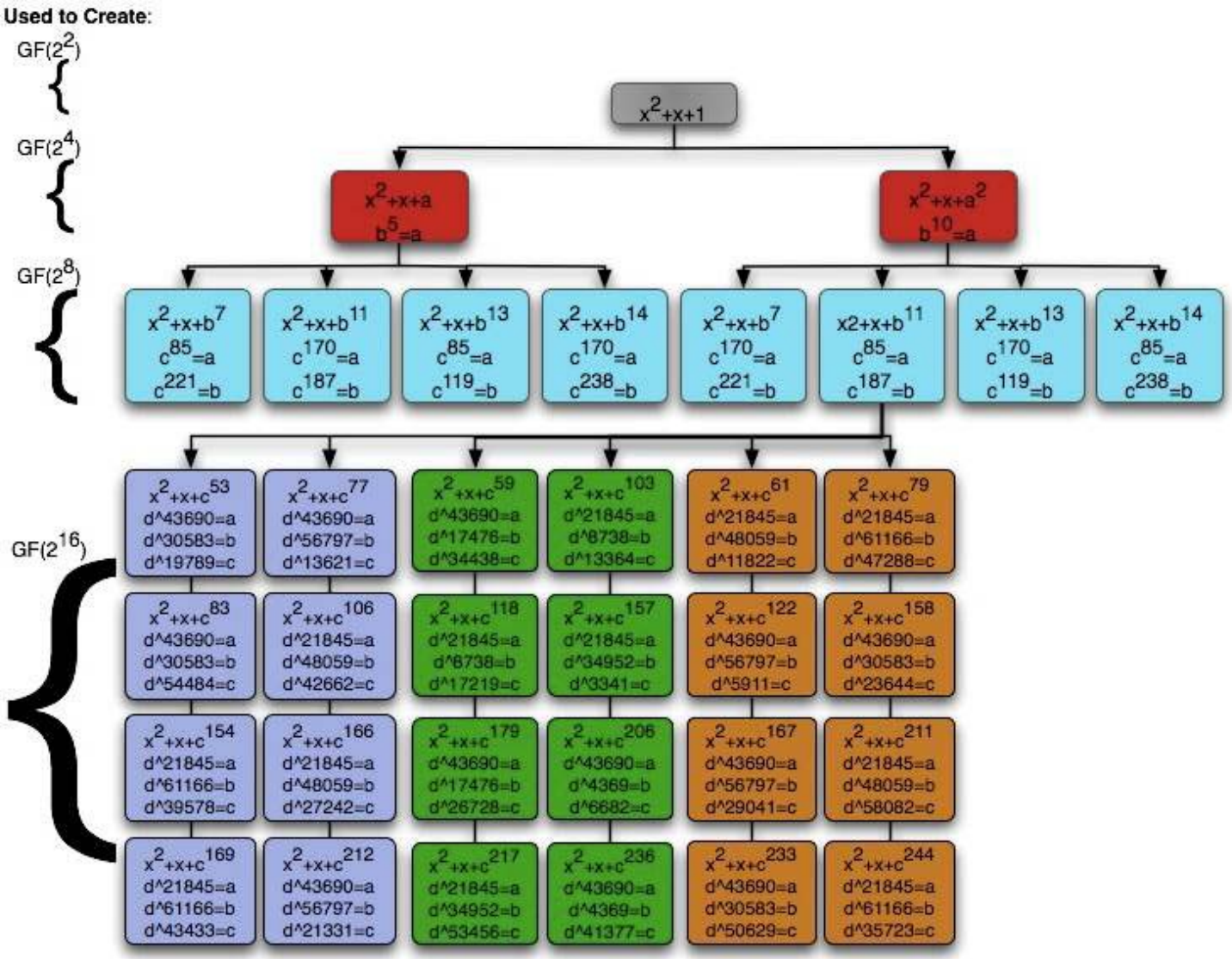

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^11
d^21845=a
d^17476=b
d^29812=c
x^2+x+c^22
d^43690=a
d^8738=b
d^14906=c
x^2+x+c^13
d^43690=a
d^54952=b
d^40349=c
x^2+x+c^26
d^21845=a
d^17476=b
d^52942=c
x^2+x+c^19
d^43690=a
d^48059=b
d^24158=c
x^2+x+c^38
d^21845=a
d^56797=b
d^12079=c
x^2+x+c^29
d^21845=a
d^4369=b
d^11308=c
x^2+x+c^58
d^43690=a
d^34952=b
d^5654=c
x^2+x+c^44
d^21845=a
d^4369=b
d^7453=c
x^2+x+c^88
d^43690=a
d^34952=b
d^36494=c
x^2+x+c^52
d^43690=a
d^8738=b
d^26471=c
x^2+x+c^67
d^43690=a
d^8738=b
d^30326=c
x^2+x+c^49
d^43690=a
d^48059=b
d^58853=c
x^2+x+c^76
d^43690=a
d^61166=b
d^38807=c
x^2+x+c^71
d^21845=a
d^17476=b
d^45232=c
x^2+x+c^116
d^21845=a
d^17476=b
d^2827=c
x^2+x+c^97
d^43690=a
d^8738=b
d^41891=c
x^2+x+c^133
d^43690=a
d^34952=b
d^59624=c
x^2+x+c^104
d^21845=a
d^4369=b
d^46003=c
x^2+x+c^134
d^21845=a
d^4369=b
d^15163=c
x^2+x+c^98
d^21845=a
d^56797=b
d^62194=c
x^2+x+c^137
d^21845=a
d^30583=b
d^48316=c
x^2+x+c^142
d^43690=a
d^8738=b
d^22616=c
x^2+x+c^163
d^43690=a
d^34952=b
d^24929=c
x^2+x+c^176
d^21845=a
d^17476=b
d^18247=c
x^2+x+c^194
d^21845=a
d^4369=b
d^53713=c
x^2+x+c^161
d^21845=a
d^17476=b
d^60652=c
x^2+x+c^208
d^43690=a
d^34952=b
d^55769=c
x^2+x+c^152
d^21845=a
d^30583=b
d^52171=c
x^2+x+c^196
d^43690=a
d^61166=b
d^31097=c
x^2+x+c^209
d^21845=a
d^4369=b
d^49858=c
x^2+x+c^232
d^43690=a
d^8738=b
d^34181=c

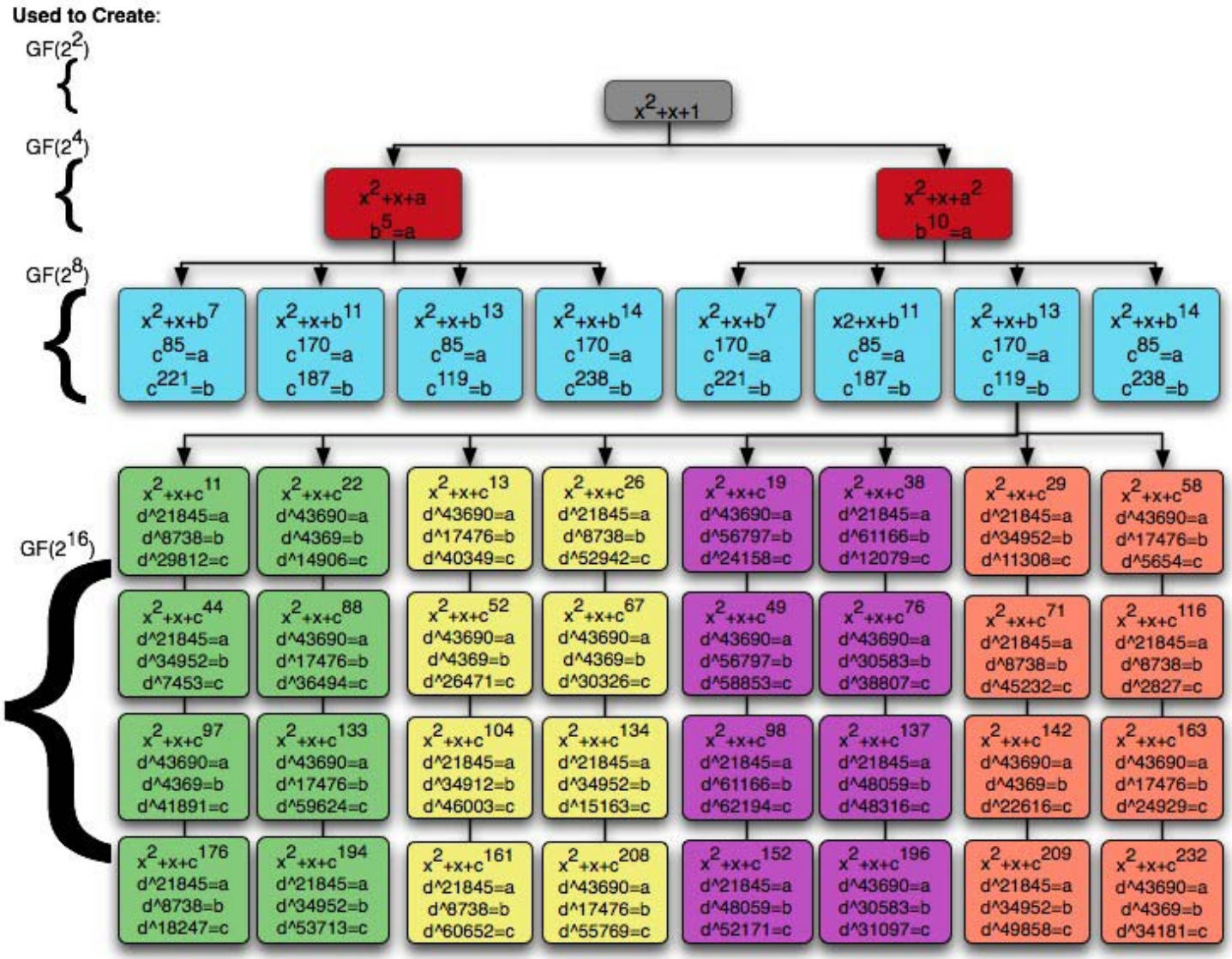

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^53
d^21845=a
d^56797=b
d^19789=c
x^2+x+c^77
d^21845=a
d^30583=b
d^13621=c
x^2+x+c^59
d^21845=a
d^4369=b
d^34438=c
x^2+x+c^103
d^43690=a
d^34952=b
d^13364=c
x^2+x+c^61
d^43690=a
d^61166=b
d^11822=c
x^2+x+c^79
d^43690=a
d^48059=b
d^47288=c
x^2+x+c^83
d^21845=a
d^56797=b
d^54484=c
x^2+x+c^106
d^43690=a
d^61166=b
d^42662=c
x^2+x+c^118
d^43690=a
d^34952=b
d^17219=c
x^2+x+c^157
d^43690=a
d^8738=b
d^3341=c
x^2+x+c^122
d^21845=a
d^30583=b
d^5911=c
x^2+x+c^158
d^21845=a
d^56797=b
d^23644=c
x^2+x+c^154
d^43690=a
d^48059=b
d^39578=c
x^2+x+c^166
d^43690=a
d^61166=b
d^27242=c
x^2+x+c^179
d^21845=a
d^4369=b
d^26728=c
x^2+x+c^206
d^21845=a
d^17476=b
d^6682=c
x^2+x+c^167
d^21845=a
d^30583=b
d^29041=c
x^2+x+c^211
d^43690=a
d^61166=b
d^58082=c
x^2+x+c^169
d^43690=a
d^48059=b
d^43433=c
x^2+x+c^212
d^21845=a
d^30583=b
d^21331=c
x^2+x+c^217
d^43690=a
d^8738=b
d^53456=c
x^2+x+c^236
d^21845=a
d^17476=b
d^41377=c
x^2+x+c^233
d^21845=a
d^56797=b
d^50629=c
x^2+x+c^244
d^43690=a
d^48059=b
d^35723=c

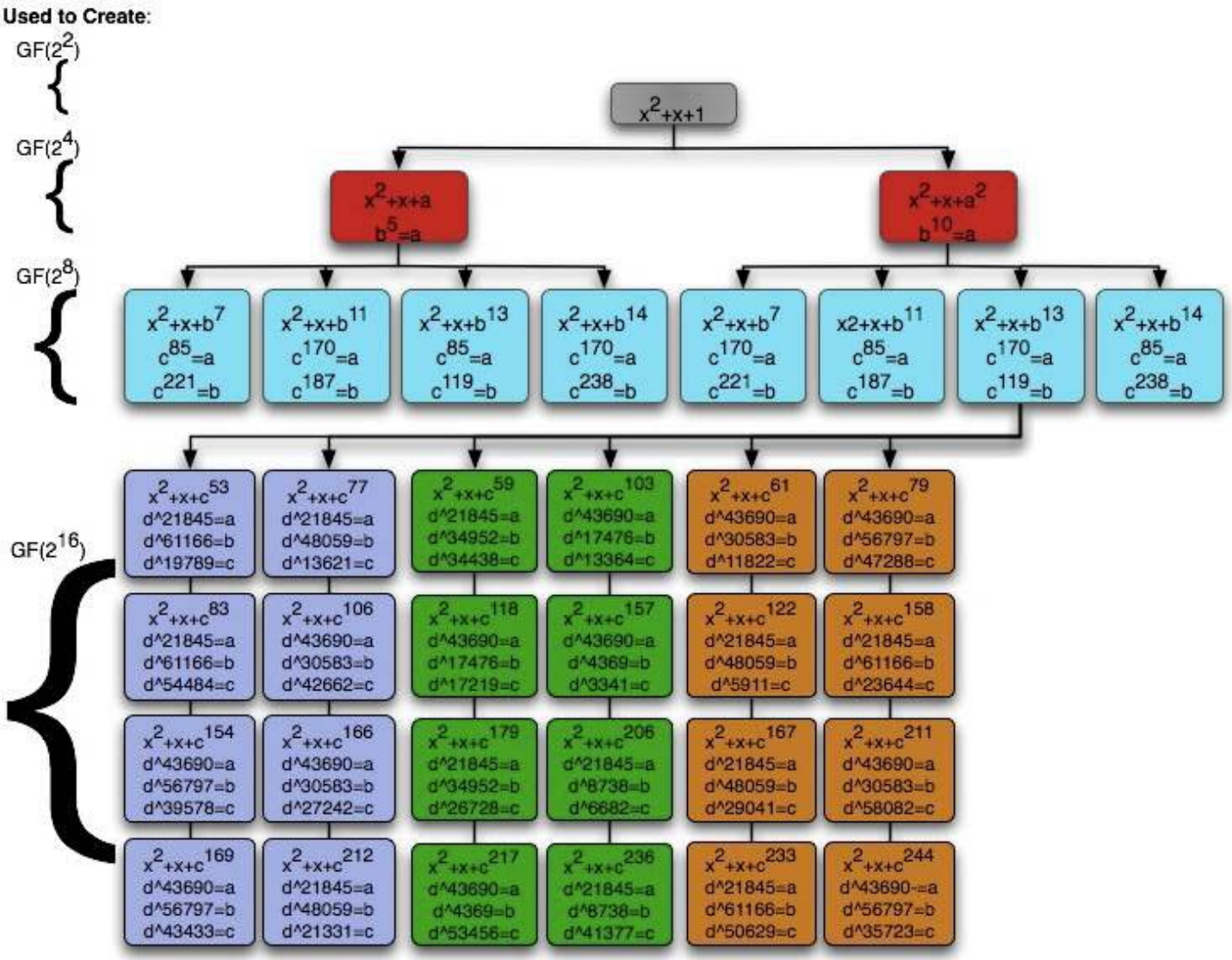

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^11
d^21845=a
d^34952=b
d^29182=c
x^2+x+c^22
d^43690=a
d^17476=b
d^14906=c
x^2+x+c^13
d^43690=a
d^4369=b
d^40349=c
x^2+x+c^26
d^21845=a
d^34952=b
d^52942=c
x^2+x+c^19
d^43690=a
d^30583=b
d^24158=c
x^2+x+c^38
d^21845=a
d^48059=b
d^12079=c
x^2+x+c^29
d^21845=a
d^8738=b
d^11308=c
x^2+x+c^58
d^43690=a
d^4369=b
d^5654=c
x^2+x+c^44
d^21845=a
d^8738=b
d^7453=c
x^2+x+c^88
d^43690=a
d^4369=b
d^36494=c
x^2+x+c^52
d^43690=a
d^17476=b
d^26471=c
x^2+x+c^67
d^43690=a
d^17476=b
d^30326=c
x^2+x+c^49
d^43690=a
d^30583=b
d^58853=c
x^2+x+c^76
d^43690=a
d^56797=b
d^38807=c
x^2+x+c^71
d^21845=a
d^34952=b
d^45232=c
x^2+x+c^116
d^21845=a
d^34952=b
d^2827=c
x^2+x+c^97
d^43690=a
d^17476=b
d^41891=c
x^2+x+c^133
d^43690=a
d^4369=b
d^59624=c
x^2+x+c^104
d^21845=a
d^8738=b
d^46003=c
x^2+x+c^134
d^21845=a
d^8738=b
d^15163=c
x^2+x+c^98
d^21845=a
d^48059=b
d^62194=c
x^2+x+c^137
d^21845=a
d^61166=b
d^48316=c
x^2+x+c^142
d^43690=a
d^17476=b
d^22616=c
x^2+x+c^163
d^43690=a
d^4369=b
d^24929=c
x^2+x+c^176
d^21845=a
d^34952=b
d^18247=c
x^2+x+c^194
d^21845=a
d^8738=b
d^53713=c
x^2+x+c^161
d^21845=a
d^34952=b
d^60652=c
x^2+x+c^208
d^43690=a
d^4369=b
d^55769=c
x^2+x+c^152
d^21845=a
d^61166=b
d^52171=c
x^2+x+c^196
d^43690=a
d^56797=b
d^31097=c
x^2+x+c^209
d^21845=a
d^8738=b
d^49858=c
x^2+x+c^232
d^43690=a
d^17476=b
d^34181=c

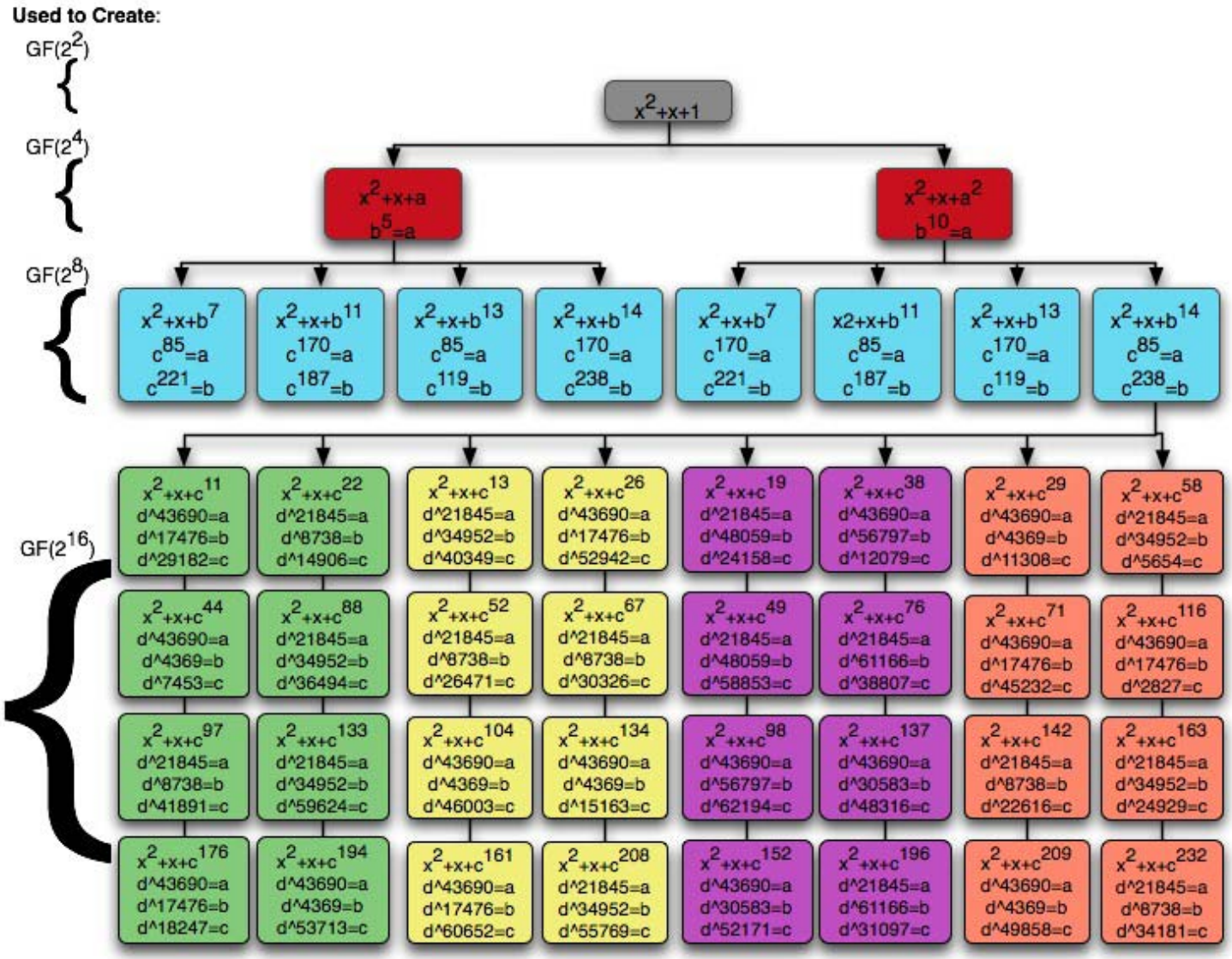

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^53
d^21845=a
d^48059=b
d^19789=c
x^2+x+c^77
d^21845=a
d^61166=b
d^13621=c
x^2+x+c^59
d^21845=a
d^8738=b
d^34438=c
x^2+x+c^103
d^43690=a
d^4369=b
d^13364=c
x^2+x+c^61
d^43690=a
d^56797=b
d^11822=c
x^2+x+c^79
d^43690=a
d^30583=b
d^47288=c
x^2+x+c^83
d^21845=a
d^48059=b
d^54484=c
x^2+x+c^106
d^43690=a
d^56797=b
d^42662=c
x^2+x+c^118
d^43690=a
d^4369=b
d^17219=c
x^2+x+c^157
d^43690=a
d^17476=b
d^3341=c
x^2+x+c^122
d^21845=a
d^61166=b
d^5911=c
x^2+x+c^158
d^21845=a
d^48059=b
d^23644=c
x^2+x+c^154
d^43690=a
d^30583=b
d^39578=c
x^2+x+c^166
d^43690=a
d^56797=b
d^27242=c
x^2+x+c^179
d^21845=a
d^8738=b
d^26728=c
x^2+x+c^206
d^21845=a
d^34952=b
d^6682=c
x^2+x+c^167
d^21845=a
d^61166=b
d^29041=c
x^2+x+c^211
d^43690=a
d^56797=b
d^58082=c
x^2+x+c^169
d^43690=a
d^30583=b
d^43433=c
x^2+x+c^212
d^21845=a
d^61166=b
d^21331=c
x^2+x+c^217
d^43690=a
d^17476=b
d^53456=c
x^2+x+c^236
d^21845=a
d^34952=b
d^41377=c
x^2+x+c^233
d^21845=a
d^48059=b
d^50629=c
x^2+x+c^244
d^43690=a
d^30583=b
d^35723=c

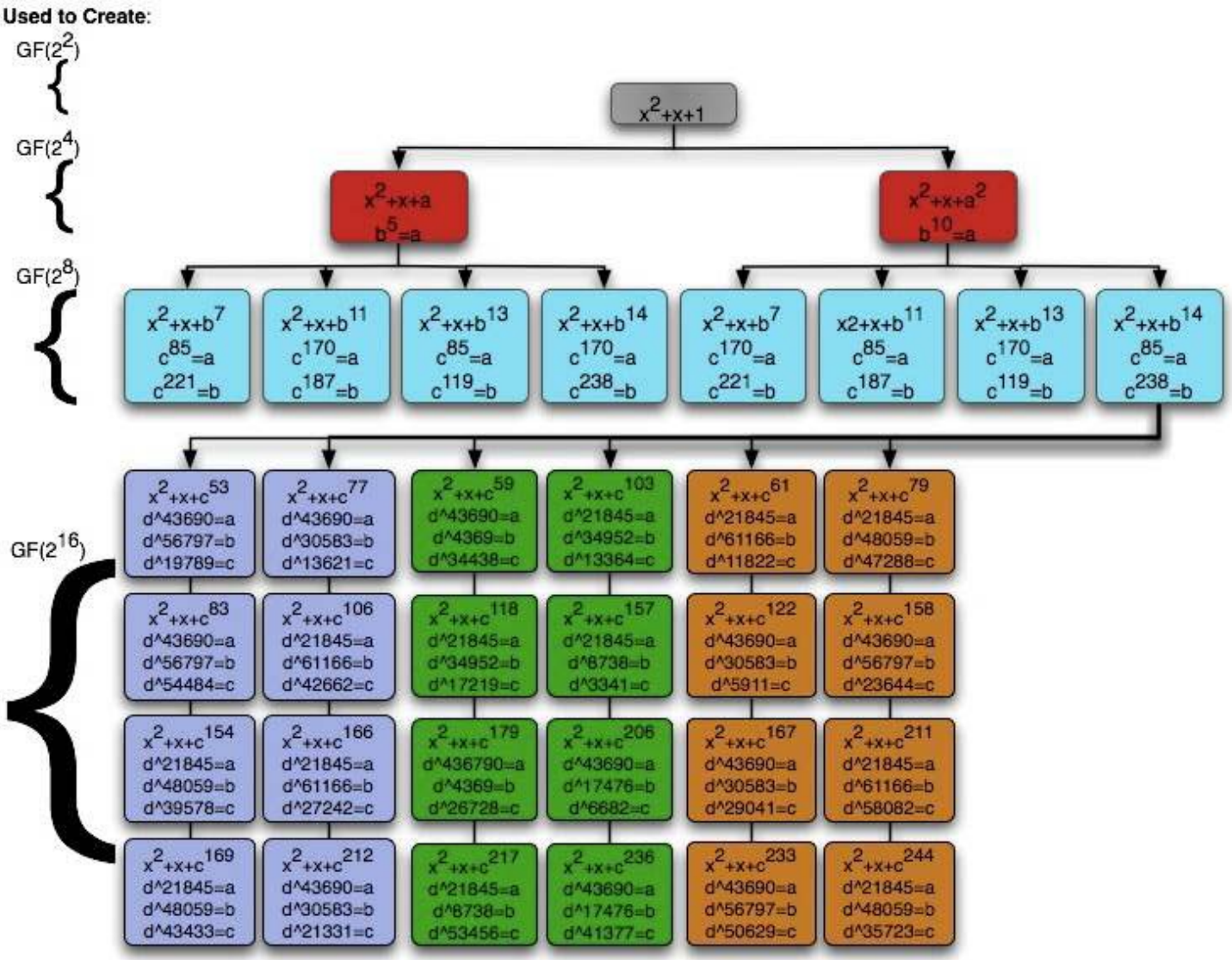

Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^11
d^43690=a
d^4369=b
d^29812=c
x^2+x+c^22
d^21845=a
d^34952=b
d^14906=c
x^2+x+c^13
d^21845=a
d^8738=b
d^40349=c
x^2+x+c^26
d^43690=a
d^4369=b
d^52942=c
x^2+x+c^19
d^21845=a
d^61166=b
d^24158=c
x^2+x+c^38
d^43690=a
d^30583=b
d^12079=c
x^2+x+c^29
d^43690=a
d^17476=b
d^11308=c
x^2+x+c^58
d^21845=a
d^8738=b
d^5654=c
x^2+x+c^44
d^43690=a
d^17476=b
d^7453=c
x^2+x+c^88
d^21845=a
d^8738=b
d^36494=c
x^2+x+c^52
d^21845=a
d^34952=b
d^26471=c
x^2+x+c^67
d^21845=a
d^34952=b
d^30326=c
x^2+x+c^49
d^21845=a
d^61166=b
d^58853=c
x^2+x+c^76
d^21845=a
d^48059=b
d^38807=c
x^2+x+c^71
d^43690=a
d^4369=b
d^45232=c
x^2+x+c^116
d^43690=a
d^4369=b
d^2827=c
x^2+x+c^97
d^21845=a
d^34952=b
d^41891=c
x^2+x+c^133
d^21845=a
d^8738=b
d^59624=c
x^2+x+c^104
d^43690=a
d^17476=b
d^46003=c
x^2+x+c^134
d^43690=a
d^17476=b
d^15163=c
x^2+x+c^98
d^43690=a
d^30583=b
d^62194=c
x^2+x+c^137
d^43690=a
d^56797=b
d^48316=c
x^2+x+c^142
d^21845=a
d^34952=b
d^22616=c
x^2+x+c^163
d^21845=a
d^8738=b
d^24929=c
x^2+x+c^176
d^43690=a
d^4369=b
d^18247=c
x^2+x+c^194
d^43690=a
d^17476=b
d^53713=c
x^2+x+c^161
d^43690=a
d^4369=b
d^60652=c
x^2+x+c^208
d^21845=a
d^8738=b
d^55769=c
x^2+x+c^152
d^43690=a
d^56797=b
d^52171=c
x^2+x+c^196
d^21845=a
d^48059=b
d^31097=c
x^2+x+c^209
d^43690=a
d^17476=b
d^49858=c
x^2+x+c^232
d^21845=a
d^34952=b
d^34181=c


Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^53
d^43690=a
d^30583=b
d^19789=c
x^2+x+c^77
d^43690=a
d^56797=b
d^13621=c
x^2+x+c^59
d^43690=a
d^17476=b
d^34438=c
x^2+x+c^103
d^21845=a
d^8738=b
d^13364=c
x^2+x+c^61
d^21845=a
d^48059=b
d^11822=c
x^2+x+c^79
d^21845=a
d^61166=b
d^47288=c
x^2+x+c^83
d^43690=a
d^30583=b
d^54484=c
x^2+x+c^106
d^21845=a
d^48059=b
d^42662=c
x^2+x+c^118
d^21845=a
d^8738=b
d^17219=c
x^2+x+c^157
d^21845=a
d^34952=b
d^3341=c
x^2+x+c^122
d^43690=a
d^56797=b
d^5911=c
x^2+x+c^158
d^43690=a
d^30583=b
d^23644=c
x^2+x+c^154
d^21845=a
d^61166=b
d^39578=c
x^2+x+c^166
d^21845=a
d^48059=b
d^27242=c
x^2+x+c^179
d^43690=a
d^17476=b
d^26728=c
x^2+x+c^206
d^43690=a
d^4369=b
d^6682=c
x^2+x+c^167
d^43690=a
d^56797=b
d^29041=c
x^2+x+c^211
d^21845=a
d^48059=b
d^58082=c
x^2+x+c^169
d^21845=a
d^61166=b
d^43433=c
x^2+x+c^212
d^43690=a
d^56797=b
d^21331=c
x^2+x+c^217
d^21845=a
d^34952=b
d^53456=c
x^2+x+c^236
d^43690=a
d^4369=b
d^41377=c
x^2+x+c^233
d^43690=a
d^30583=b
d^50629=c
x^2+x+c^244
d^21845=a
d^61166=b
d^35723=c


Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^11
d^21845=a
d^8738=b
d^29812=c
x^2+x+c^22
d^43690=a
d^4369=b
d^14906=c
x^2+x+c^13
d^43690=a
d^17476=b
d^40349=c
x^2+x+c^26
d^21845=a
d^8738=b
d^52942=c
x^2+x+c^19
d^43690=a
d^56797=b
d^24158=c
x^2+x+c^38
d^21845=a
d^61166=b
d^12079=c
x^2+x+c^29
d^21845=a
d^34952=b
d^11308=c
x^2+x+c^58
d^43690=a
d^17476=b
d^5654=c
x^2+x+c^44
d^21845=a
d^34952=b
d^7453=c
x^2+x+c^88
d^43690=a
d^17476=b
d^36494=c
x^2+x+c^52
d^43690=a
d^4369=b
d^26471=c
x^2+x+c^67
d^43690=a
d^4369=b
d^30326=c
x^2+x+c^49
d^43690=a
d^56797=b
d^58853=c
x^2+x+c^76
d^43690=a
d^30583=b
d^38807=c
x^2+x+c^71
d^21845=a
d^8738=b
d^45232=c
x^2+x+c^116
d^21845=a
d^8738=b
d^2827=c
x^2+x+c^97
d^43690=a
d^4369=b
d^41891=c
x^2+x+c^133
d^43690=a
d^17476=b
d^59624=c
x^2+x+c^104
d^21845=a
d^34912=b
d^46003=c
x^2+x+c^134
d^21845=a
d^34952=b
d^15163=c
x^2+x+c^98
d^21845=a
d^61166=b
d^62194=c
x^2+x+c^137
d^21845=a
d^48059=b
d^48316=c
x^2+x+c^142
d^43690=a
d^4369=b
d^22616=c
x^2+x+c^163
d^43690=a
d^17476=b
d^24929=c
x^2+x+c^176
d^21845=a
d^8738=b
d^18247=c
x^2+x+c^194
d^21845=a
d^34952=b
d^53713=c
x^2+x+c^161
d^21845=a
d^8738=b
d^60652=c
x^2+x+c^208
d^43690=a
d^17476=b
d^55769=c
x^2+x+c^152
d^21845=a
d^48059=b
d^52171=c
x^2+x+c^196
d^43690=a
d^30583=b
d^31097=c
x^2+x+c^209
d^21845=a
d^34952=b
d^49858=c
x^2+x+c^232
d^43690=a
d^4369=b
d^34181=c


Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^53
d^21845=a
d^61166=b
d^19789=c
x^2+x+c^77
d^21845=a
d^48059=b
d^13621=c
x^2+x+c^59
d^21845=a
d^34952=b
d^34438=c
x^2+x+c^103
d^43690=a
d^17476=b
d^13364=c
x^2+x+c^61
d^43690=a
d^30583=b
d^11822=c
x^2+x+c^79
d^43690=a
d^56797=b
d^47288=c
x^2+x+c^83
d^21845=a
d^61166=b
d^54484=c
x^2+x+c^106
d^43690=a
d^30583=b
d^42662=c
x^2+x+c^118
d^43690=a
d^17476=b
d^17219=c
x^2+x+c^157
d^43690=a
d^4369=b
d^3341=c
x^2+x+c^122
d^21845=a
d^48059=b
d^5911=c
x^2+x+c^158
d^21845=a
d^61166=b
d^23644=c
x^2+x+c^154
d^43690=a
d^56797=b
d^39578=c
x^2+x+c^166
d^43690=a
d^30583=b
d^27242=c
x^2+x+c^179
d^21845=a
d^34952=b
d^26728=c
x^2+x+c^206
d^21845=a
d^8738=b
d^6682=c
x^2+x+c^167
d^21845=a
d^48059=b
d^29041=c
x^2+x+c^211
d^43690=a
d^30583=b
d^58082=c
x^2+x+c^169
d^43690=a
d^56797=b
d^43433=c
x^2+x+c^212
d^21845=a
d^48059=b
d^21331=c
x^2+x+c^217
d^43690=a
d^4369=b
d^53456=c
x^2+x+c^236
d^21845=a
d^8738=b
d^41377=c
x^2+x+c^233
d^21845=a
d^61166=b
d^50629=c
x^2+x+c^244
d^43690=a
d^56797=b
d^35723=c


Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^11
d^43690=a
d^17476=b
d^29182=c
x^2+x+c^22
d^21845=a
d^8738=b
d^14906=c
x^2+x+c^13
d^21845=a
d^34952=b
d^40349=c
x^2+x+c^26
d^43690=a
d^17476=b
d^52942=c
x^2+x+c^19
d^21845=a
d^48059=b
d^24158=c
x^2+x+c^38
d^43690=a
d^56797=b
d^12079=c
x^2+x+c^29
d^43690=a
d^4369=b
d^11308=c
x^2+x+c^58
d^21845=a
d^34952=b
d^5654=c
x^2+x+c^44
d^43690=a
d^4369=b
d^7453=c
x^2+x+c^88
d^21845=a
d^34952=b
d^36494=c
x^2+x+c^52
d^21845=a
d^8738=b
d^26471=c
x^2+x+c^67
d^21845=a
d^8738=b
d^30326=c
x^2+x+c^49
d^21845=a
d^48059=b
d^58853=c
x^2+x+c^76
d^21845=a
d^61166=b
d^38807=c
x^2+x+c^71
d^43690=a
d^17476=b
d^45232=c
x^2+x+c^116
d^43690=a
d^17476=b
d^2827=c
x^2+x+c^97
d^21845=a
d^8738=b
d^41891=c
x^2+x+c^133
d^21845=a
d^34952=b
d^59624=c
x^2+x+c^104
d^43690=a
d^4369=b
d^46003=c
x^2+x+c^134
d^43690=a
d^4369=b
d^15163=c
x^2+x+c^98
d^43690=a
d^56797=b
d^62194=c
x^2+x+c^137
d^43690=a
d^30583=b
d^48316=c
x^2+x+c^142
d^21845=a
d^8738=b
d^22616=c
x^2+x+c^163
d^21845=a
d^34952=b
d^24929=c
x^2+x+c^176
d^43690=a
d^17476=b
d^18247=c
x^2+x+c^194
d^43690=a
d^4369=b
d^53713=c
x^2+x+c^161
d^43690=a
d^17476=b
d^60652=c
x^2+x+c^208
d^21845=a
d^34952=b
d^55769=c
x^2+x+c^152
d^43690=a
d^30583=b
d^52171=c
x^2+x+c^196
d^21845=a
d^61166=b
d^31097=c
x^2+x+c^209
d^43690=a
d^4369=b
d^49858=c
x^2+x+c^232
d^21845=a
d^8738=b
d^34181=c


Used to Create:
GF(2^2)
GF(2^4)
GF(2^8)
GF(2^16)
x^2+x+1
x^2+x+a
b^5=a
x^2+x+a^2
b^10=a
x^2+x+b^7
c^85=a
c^221=b
x^2+x+b^11
c^170=a
c^187=b
x^2+x+b^13
c^85=a
c^119=b
x^2+x+b^14
c^170=a
c^238=b
x^2+x+b^7
c^170=a
c^221=b
x2+x+b^11
c^85=a
c^187=b
x^2+x+b^13
c^170=a
c^119=b
x^2+x+b^14
c^85=a
c^238=b
x^2+x+c^53
d^43690=a
d^56797=b
d^19789=c
x^2+x+c^77
d^43690=a
d^30583=b
d^13621=c
x^2+x+c^59
d^43690=a
d^4369=b
d^34438=c
x^2+x+c^103
d^21845=a
d^34952=b
d^13364=c
x^2+x+c^61
d^21845=a
d^61166=b
d^11822=c
x^2+x+c^79
d^21845=a
d^48059=b
d^47288=c
x^2+x+c^83
d^43690=a
d^56797=b
d^54484=c
x^2+x+c^106
d^21845=a
d^61166=b
d^42662=c
x^2+x+c^118
d^21845=a
d^34952=b
d^17219=c
x^2+x+c^157
d^21845=a
d^8738=b
d^3341=c
x^2+x+c^122
d^43690=a
d^30583=b
d^5911=c
x^2+x+c^158
d^43690=a
d^56797=b
d^23644=c
x^2+x+c^154
d^21845=a
d^48059=b
d^39578=c
x^2+x+c^166
d^21845=a
d^61166=b
d^27242=c
x^2+x+c^179
d^436790=a
d^4369=b
d^26728=c
x^2+x+c^206
d^43690=a
d^17476=b
d^6682=c
x^2+x+c^167
d^43690=a
d^30583=b
d^29041=c
x^2+x+c^211
d^21845=a
d^61166=b
d^58082=c
x^2+x+c^169
d^21845=a
d^48059=b
d^43433=c
x^2+x+c^212
d^43690=a
d^30583=b
d^21331=c
x^2+x+c^217
d^21845=a
d^8738=b
d^53456=c
x^2+x+c^236
d^43690=a
d^17476=b
d^41377=c
x^2+x+c^233
d^43690=a
d^56797=b
d^50629=c
x^2+x+c^244
d^21845=a
d^48059=b
d^35723=c

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Canright, D. A Very Compact Rijndael S-box. Monterey: Naval Postgraduate School; 2004.

2. Conway, JH. *On Numbers and Games.* 2nd ed. Wellesley: AK Peters, Ltd.; 2001.

3. Dummit, D., Foote R. *Abstract Algebra.* 2nd ed. New York: Wiley; 1999.

4. Gallian, J. *Contemporary Abstract Algebra.* 4th ed. Boston: Houghton Mifflin; 1998.

5. Golomb, S. *Shift Register Sequences.* Rev ed. Laguna Hills: Aegean Park Press; 1982.

6. Fellin, JA. Primitive Shift Registers. *UMAP Module*. 1981; Unit 310.

7. Schneier, B. *Applied Cryptography.* 2nd ed. New York: Wiley; 1996.

8. Swallow, J. AlgFields.txt. Davidson: 2005; 1.02. Available from: http://www.davidson.edu/academic/math/swallow/AlgFieldsWeb/downloads.htm. Accessed February 22, 2006.

9. Fredricksen, H. Cryptography Class Notes. Naval Postgraduate School; 2006.

10. Fredricksen, H. Goldstein D. Email Correspondence. 2006.

THIS PAGE INTENTIONALLY LEFT BLANK

BIBLIOGRAPHY

Daemen, J., Rijmen, V. The Design of Rijndael. Berlin: Springer; 2002.

Fredricksen, H., Krahn, G. Determining the Generator of a Linear Recursive Sequence. Monterey: Naval Postgraduate School; 1994.

National Institute of Standards and Technology (NIST). Specifications for the Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST); 2001;Technical Report FIPS PUB 197. Accessed September 5, 2006.Available from: http://csrc.nist.gov/CryptoToolkit/aes/rijndael/.

Stinson, D. Cryptography: Theory and Practice. 3rd ed. Boca Raton: Chapman & Hall; 2006.

Swallow, J. Exploratory Galois Theory. Cambridge: Cambridge University Press; 2004.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. George Dinolt
   Naval Postgraduate School
   Monterey, California

4. Dr. Harold Fredricksen
   Naval Postgraduate School
   Monterey, California

5. Jody Radowicz
   Naval Postgraduate School
   Monterey, California